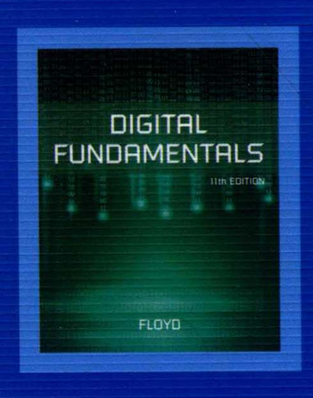
国外电子与通信教材系列

Digital Fundamentals, Eleventh Edition



数字电子技术（第十一版）（英文版）

[美] Thomas L. Floyd 著

余 璆　熊 洁　改编

中国工信出版集团

数字电子技术（第十一版）（英文版）

Digital Fundamentals, Eleventh Edition

本书是一本关于数字电子技术的经典教材，并专门针对国内教学的实际情况进行了缩减。全书主要介绍了数字电子技术的基本概念、数制、逻辑门、布尔代数和逻辑化简、组合逻辑分析、组合逻辑的作用、计数器、移位寄存器、存储器、可编程逻辑与软件、数据传输、集成电路技术等。全书的特色在于示例与习题丰富、图解清晰、语言流畅、写作风格简约。

新版特色：

- 全新的页面布局和设计
- 删除了前几版过时的设备
- 新增了布尔简化的Q-M方法
- 新增了Moore和Mealy状态机的内容
- 新增了有关数据传输的一章，包括标准总线的全面介绍
- 突出了D触发器的使用

本书可向授课教师提供教辅（习题解答，PPT等），具体申请方式请参见书后的教辅申请说明。

改编者简介／余 璆，熊 洁

上海工程技术大学教师，长期负责电子技术课程的教学工作。

其他相关图书
请扫二维码↓

定价：79.00 元

策划编辑：冯小贝
责任编辑：冯小贝
责任美编：孙焱津

国外电子与通信教材系列

数字电子技术

（第十一版）（英文版）

Digital Fundamentals

Eleventh Edition

［美］Thomas L. Floyd　著

余　璆　熊　洁　改编

電子工業出版社
Publishing House of Electronics Industry
北京·BEIJING

内 容 简 介

本书是一本关于数字电子技术的经典教材，并专门针对国内教学的实际情况进行了缩减。全书主要介绍了数字电子技术的基本概念、数制、逻辑门、布尔代数和逻辑化简、组合逻辑分析、组合逻辑的作用、计数器、移位寄存器、存储器、可编程逻辑与软件、数据传输、集成电路技术等。全书的特色在于示例与习题丰富、图解清晰、语言流畅、写作风格简约。

本书可作为高等院校电子信息类相关专业本科生的数字电子技术课程的双语教材，也可供相关技术、科研人员使用，或作为继续教育的参考用书。

图书在版编目(CIP)数据

数字电子技术：第十一版=Digital Fundamentals，Eleventh Edition：英文/（美）托马斯·L. 弗洛伊德（Thomas L. Floyd）著；余瓔，熊洁改编. —北京：电子工业出版社，2017.8
国外电子与通信教材系列
ISBN 978-7-121-31982-2

Ⅰ. ①数… Ⅱ. ①托… ②余… ③熊… Ⅲ. ①数字电路－电子技术－高等学校－教材－英文 Ⅳ. ①TN79

中国版本图书馆CIP数据核字（2017）第139781号

策划编辑：冯小贝
责任编辑：冯小贝
印　　刷：三河市良远印务有限公司
装　　订：三河市良远印务有限公司
出版发行：电子工业出版社
　　　　　北京市海淀区万寿路173信箱　邮编：100036
开　　本：787×1092　1/16　印张：31.5　字数：1179千字
版　　次：2006年6月第1版（原著第9版）
　　　　　2017年8月第3版（原著第11版）
印　　次：2017年8月第1次印刷
定　　价：79.00元

前　言

Digital Fundamentals（即改编后的《数字电子技术》）的第十一版继续秉承长期以来的传统，侧重于数字技术核心的基础内容。本书通过丰富的插图、举例、练习和若干应用帮助读深入理解相关的基本概念。除此以外，本书也涉及了应用逻辑功能、功能的实现、集成电路技术，还涉及了信号转换和处理、数据传输、处理和控制等一些特定主题。第十一版增加了一些新的内容和特点，许多原有的内容也得到了扩展。

书中涉及的知识旨在让学生进一步学习更高层次或选修内容之前，掌握所有重要的基础概念。涉及的内容范围具有灵活性，以便适应各种课程的需求。例如，对于某些课程并不太适合的设计性课题或者应用性课题，忽略或减少相关的内容不会影响书中基本概念的学习。半导体电路的背景知识并非本书必需的先修内容，集成电路技术（芯片内部电路）的知识则有选择地提及一些。

第十一版的新内容

- 全新的页面布局和设计，视觉效果更好，容易使用，修改和改进了一些内容。
- 删除了过时的设备。
- 新增了布尔简化的Q-M（Quine-McCluskey）方法。
- 新增了Moore和Mealy状态机的内容。
- 新增了有关数据传输的一章，包括标准总线的全面介绍。
- 突出了D触发器的使用。

本书的特性

- 基本核心内容和高层次的或外围的内容不相混合。
- *InfoNotes*（计算机小知识）以侧边栏的版面形式给出精炼的趣味短文。
- 每章都提示学生如何找到各种练习的答案。
- 每章中的各个小节都有检查题，答案列在每章的最后。
- 给出的每个例题都带有相关的练习，答案列在每章的最后。
- 分散在各处的*Hands-On Tips*（实践技巧）提供有用的实践知识。
- 网站上的MultiSim文件提供书中选做的仿真参考电路。
- 每章最后的是非判断题。
- 每章最后的单选自测题。
- 每章最后的分节习题在本书的最后给出奇数题目的答案。

学生资源[1]

- 可选择购买的实验手册（*Experiments in Digital Fundamentals, Eleventh Edition*），由Dave Buchla 和 Doug Joksch 撰写。
- *MultiSim*电路。网站（www.pearsonhighered.com/careersresources.com）上的MultiSim文件，包含了书中选做的仿真参考电路，由图P-1表示。

MultiSim

图 P-1

- 在线章节，"Intergrated Circuit Technologies"
- VHDL教程
- Verilog教程
- MultiSim教程
- Altera Quartus II教程
- Xilinx ISE教程
- 变量卡诺图教程
- 汉明码教程
- Q-M方法教程，等等

教师资源[2]

- 教师资源手册，包含相关章节的习题解答，应用逻辑习题的解答，MultiSim仿真结果的总结，由Dave Buchla 和 Doug Joksch 撰写的实验手册的实验结果。
- 书中插图的幻灯片文件。

一些专题的说明

检查题 每个小节的结尾都有习题组成的复习部分，以加强这一小节主要概念的理解。这个特点如图P-2所示。

SECTION 5–1 CHECKUP

Answers are at the end of the chapter.

1. Determine the output (1 or 0) of a 4-variable AND-OR-Invert circuit for each of the following input conditions:
   (a) $A = 1, B = 0, C = 1, D = 0$ (b) $A = 1, B = 1, C = 0, D = 1$
   (c) $A = 0, B = 1, C = 1, D = 1$
2. Determine the output (1 or 0) of an exclusive-OR gate for each of the following input conditions:
   (a) $A = 1, B = 0$ (b) $A = 1, B = 1$
   (c) $A = 0, B = 1$ (d) $A = 0, B = 0$
3. Develop the truth table for a certain 3-input logic circuit with the output expression $X = A\overline{B}C + \overline{A}BC + \overline{A}\overline{B}\overline{C} + AB\overline{C} + ABC$.
4. Draw the logic diagram for an exclusive-NOR circuit.

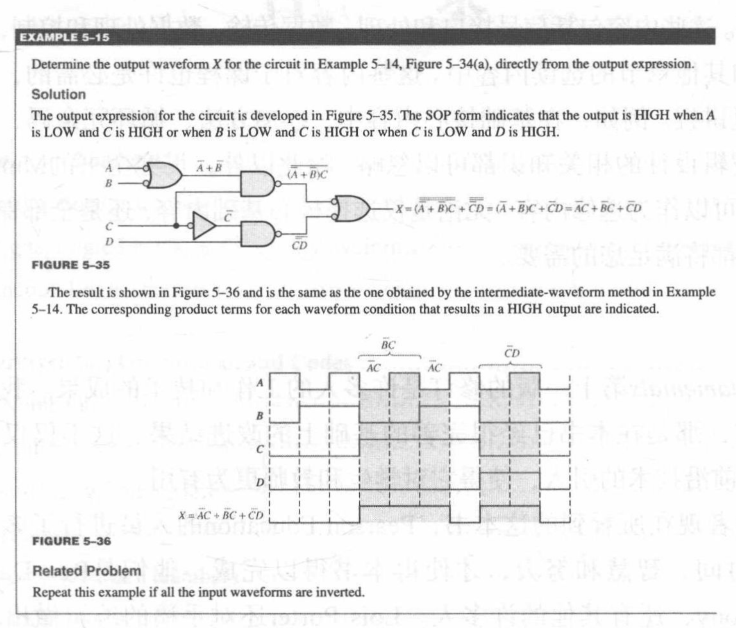
图 P-2

检查题的答案列在本章的结尾。

① 相关的一些资源也可登录华信教育资源网 www.hxedu.com.cn 注册下载。

② 教师资源申请方式请参见书后的"教辅申请表"。

例题和相关的问题 书中给出了丰富的例题，用以帮助对基本概念进行解释。每个例题都配有相关的问题，通过让学生解答和例题相似的题目来加强和拓宽所学知识。典型的例题和相关问题如图P-3所示。



EXAMPLE 5–15

Determine the output waveform *X* for the circuit in Example 5–14, Figure 5–34(a), directly from the output expression.

Solution

The output expression for the circuit is developed in Figure 5–35. The SOP form indicates that the output is HIGH when *A* is LOW and *C* is HIGH or when *B* is LOW and *C* is HIGH or when *C* is LOW and *D* is HIGH.

FIGURE 5–35

The result is shown in Figure 5–36 and is the same as the one obtained by the intermediate-waveform method in Example 5–14. The corresponding product terms for each waveform condition that results in a HIGH output are indicated.

FIGURE 5–36

Related Problem

Repeat this example if all the input waveforms are inverted.

图 P-3

献给学生

数字技术遍布于我们生活的方方面面。例如手机和其他类型的无线通信，包括电视、收音机、过程控制、汽车电子、消费电子、飞机导航等，本书仅仅给出了为数不多的依赖于数字电子的应用。具有坚实的数字技术基础知识，会使您在将来得到技术含量很高的工作。最重要的事情，就是理解数字基础的核心内容，这样就可以深入到其他内容的学习。

献给教师

通常，时间的限制或课程的侧重点确定了本课程涉及的内容。为了满足特定课程的特定内容，忽略或强调一些知识及改变某些内容的学习顺序是很常见的。本教材为授课内容的选取提供了很大的灵活性。某些相关主题分散在不同的章节，如果忽略了某些知识点，则其他的内容不会受到影响。同样，如果增加了某些知识点，它们和其余的内容则可以无缝衔接。本书围绕数字基础的核心内容编写，大部分内容对于数字课程来说都是必不可少的。围绕本课程，可以增加或减少书中的内容，取决于课程的侧重点和/或其他因素。即使是核心内容，也可以忽略一些选修的章节。

- **核心基础内容。** 有关数字技术基础的内容贯穿全书。对于围绕核心内容的一些其他主题，依据课程的需要可以增加或删除。本书涉及的内容在数字技术中是很重要的，但是围绕核心内容的每块主题可以根据特定的需要删除，这不会影响核心的基础内容的学习。

- **集成电路技术**。本书提供一章在线章节“Intergrated Circuit Technologies”。如果想讨论电路的详细特性，可以选读这一章的部分或全部内容，以补充数字基础电路的知识。忽略这一章不会影响本书其他内容的学习。
- **特殊主题**。这些内容包括信号接口和处理、数据传输、数据处理和控制，分散在第11章、第12章和其他章节的选读内容中。这些内容对于课程也许是必需的，或者也可以在其他课程里讲授。例如，在基础核心内容中，Q-M方法、循环冗余码、超前进位加法器或时序逻辑设计的相关知识都可以忽略。除此以外，贯穿全书的MultiSim（计算机仿真）内容可以作为选修内容。无论是仅选择核心基础内容，还是全部保留或介于两者之间，本书都将满足您的需要。

致谢

*Digital Fundamentals*第十一版的修订是许多人的工作和技术的成果。我认为我们已经完成了预期的工作，那是在本书已经很完善的基础上的改进结果。这不仅仅是基础内容的改善，还是最新和前沿技术的引入，使得它对学生和教师更为有用。

为了编写读者现在所看到的这本书，Pearson Education的人员进行了多个阶段的工作，倾注了大量的时间、智慧和努力，才使得本书得以完成，他们是Rex Davidson、Lindsey Gill和Vern Anthony，还有其他的许多人。Lois Porter还对手稿的编辑做出了卓越的贡献。Doug Joksch给出了VHDL编程（参阅在线资源）的相关内容。Gary Snyder修订和升级了MultiSim电路文件（参阅在线资源）。对于所有这些人和其他间接参与本书编辑的人，在此表示我的感谢和感激之情。

在修订本书和所有其他版本中，依靠了许多读者和专家的帮助。在此衷心感谢如下的审稿人，他们提出了许多有价值的建议和建设性的意见：

Dr. Cuiling Gong，Texas Christian大学

Jonathan White，Harding大学

Zane Gastineau，Harding大学

Dr. Eric Bothur，Midlands技术学院

同样感谢Pearson Education所有销售人员的努力，是他们的帮助使得本书得以和广大读者见面。此外，感谢所有选用本书作为教材的教师和个人读者。我希望您会发现*Digital Fundamentals*的第十一版比以前的版本更好，它将继续是学生学习的有价值的教材和参考资料。

Tom Floyd

目　录

① 可登录华信教育资源网（www.hxcdu.com.cn) 注册下载。

Chapter 1 Introductory Concepts

CHAPTER OUTLINE

1–1 Digital and Analog Quantities

An **analog** quantity is one having continuous values. A **digital** quantity is one having a discrete set of values. Most things that can be measured quantitatively occur in nature in analog form. For example, the air temperature changes over a continuous range of values. During a given day, the temperature does not go from, say, 70° to 71° instantaneously; it takes on all the infinite values in between. If you graphed the temperature on a typical summer day, you would have a smooth, continuous curve similar to the curve in Figure 1–1. Other examples of analog quantities are time, pressure, distance, and sound.

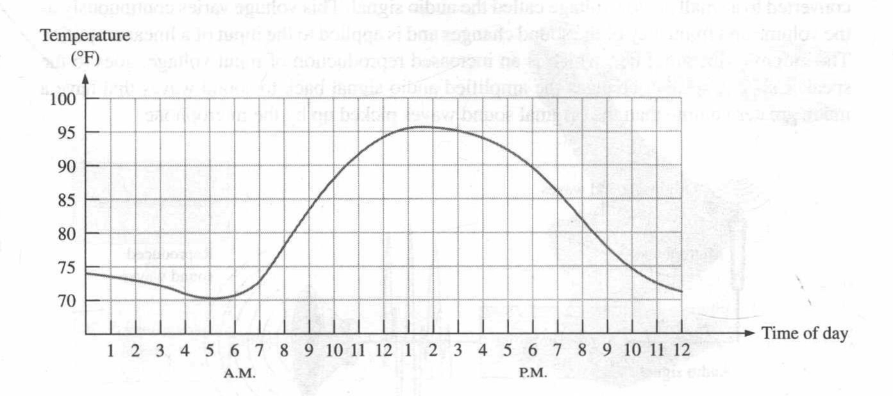


FIGURE 1–1
Graph of an analog quantity (temperature versus time).

Rather than graphing the temperature on a continuous basis, suppose you just take a temperature reading every hour. Now you have sampled values representing the temperature at discrete points in time (every hour) over a 24-hour period, as indicated in Figure 1–2. You have effectively converted an analog quantity to a form that can now be digitized by representing each sampled value by a digital code. It is important to realize that Figure 1–2 itself is not the digital representation of the analog quantity.

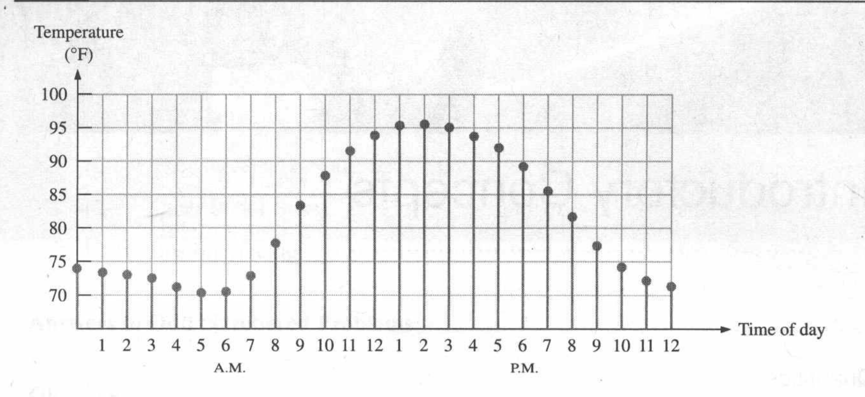


FIGURE 1–2 Sampled-value representation (quantization) of the analog quantity in Figure 1–1. Each value represented by a dot can be digitized by representing it as a digital code that consists of a series of 1s and 0s.

The Digital Advantage

Digital representation has certain advantages over analog representation in electronics applications. For one thing, digital data can be processed and transmitted more efficiently and reliably than analog data. Also, digital data has a great advantage when storage is necessary. For example, music when converted to digital form can be stored more compactly and reproduced with greater accuracy and clarity than is possible when it is in analog form. Noise (unwanted voltage fluctuations) does not affect digital data nearly as much as it does analog signals.

An Analog System

A public address system, used to amplify sound so that it can be heard by a large audience, is one simple example of an application of analog electronics. The basic diagram in Figure 1–3 illustrates that sound waves, which are analog in nature, are picked up by a microphone and converted to a small analog voltage called the audio signal. This voltage varies continuously as the volume and frequency of the sound changes and is applied to the input of a linear amplifier. The output of the amplifier, which is an increased reproduction of input voltage, goes to the speaker(s). The speaker changes the amplified audio signal back to sound waves that have a much greater volume than the original sound waves picked up by the microphone.

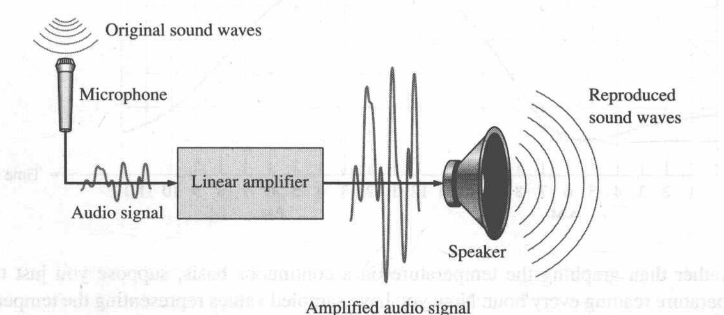


FIGURE 1–3 A basic audio public address system.

A System Using Digital and Analog Methods

The compact disk (CD) player is an example of a system in which both digital and analog circuits are used. The simplified block diagram in Figure 1–4 illustrates the basic principle. Music in digital form is stored on the compact disk. A laser diode optical system picks up the digital data from the rotating disk and transfers it to the **digital-to-analog converter (DAC)**. The DAC changes the digital data into an analog signal that is an electrical reproduction of the original music. This signal is amplified and sent to the speaker for you to

enjoy. When the music was originally recorded on the CD, a process, essentially the reverse of the one described here, using an **analog-to-digital converter (ADC)** was used.

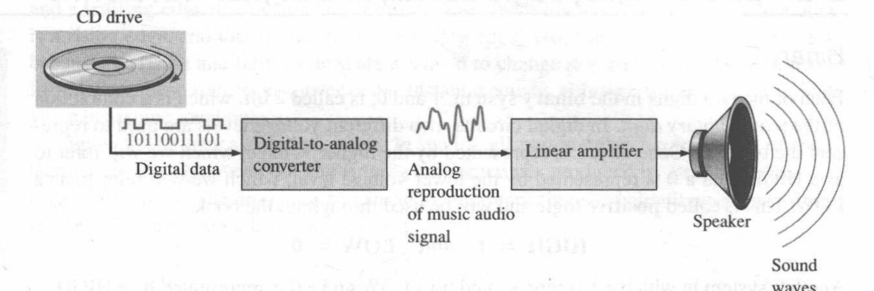


FIGURE 1–4
Basic block diagram of a CD player. Only one channel is shown.

Mechatronics

Both digital and analog electronics are used in the control of various mechanical systems. The interdisciplinary field that comprises both mechanical and electronic components is known as **mechatronics**.

Mechatronic systems are found in homes, industry, and transportation. Most home appliances consist of both mechanical and electronic components. Electronics controls the operation of a washing machine in terms of water flow, temperature, and type of cycle. Manufacturing industries rely heavily on mechatronics for process control and assembly. In automotive and other types of manufacturing, robotic arms perform precision welding, painting, and other functions on the assembly line. Automobiles themselves are mechatronic machines; a digital computer controls functions such as braking, engine parameters, fuel flow, safety features, and monitoring.

Figure 1–5(a) is a basic block diagram of a mechatronic system. A simple robotic arm is shown in Figure 1–5(b), and robotic arms on an automotive assembly line are shown in part (c).

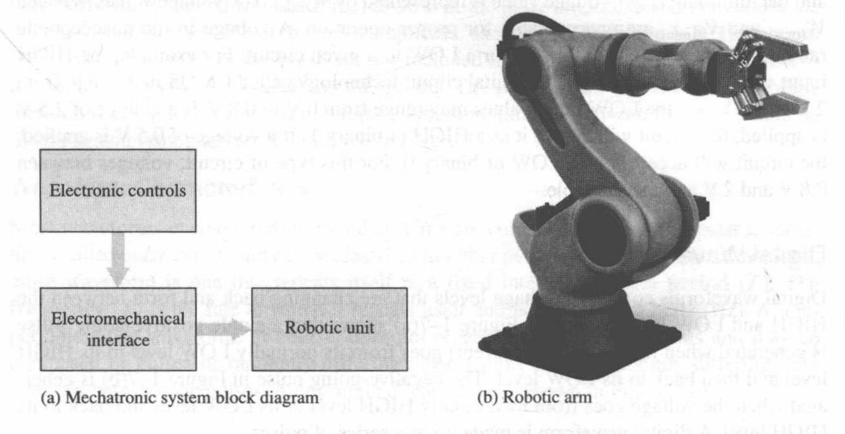


FIGURE 1–5
Example of a mechatronic system and application.

The movement of the arm in any quadrant and to any specified position is accomplished with some type of digital control such as a microcontroller.

SECTION 1-1 CHECKUP

Answers are at the end of the chapter.

1. Define *analog*.
2. Define *digital*.
3. Explain the difference between a digital quantity and an analog quantity.
4. Give an example of a system that is analog and one that is a combination of both digital and analog. Name a system that is entirely digital.
5. What does a mechatronic system consist of?

1–2 Binary Digits, Logic Levels, and Digital Waveforms

Binary Digits

Each of the two digits in the **binary** system, 1 and 0, is called a **bit**, which is a contraction of the words *binary digit.* In digital circuits, two different voltage levels are used to represent the two bits. Generally, 1 is represented by the higher voltage, which we will refer to as a HIGH, and a 0 is represented by the lower voltage level, which we will refer to as a LOW. This is called **positive logic** and will be used throughout the book.

HIGH = 1 and LOW = 0

Another system in which a 1 is represented by a LOW and a 0 is represented by a HIGH is called *negative logic.*

Groups of bits (combinations of 1s and 0s), called *codes,* are used to represent numbers, letters, symbols, instructions, and anything else required in a given application.

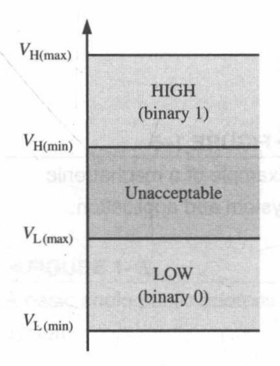
InfoNote

The concept of a digital computer can be traced back to Charles Babbage, who developed a crude mechanical computation device in the 1830s. John Atanasoff was the first to apply electronic processing to digital computing in 1939. In 1946, an electronic digital computer called ENIAC was implemented with vacuum-tube circuits. Even though it took up an entire room, ENIAC didn't have the computing power of your handheld calculator.

Logic Levels

The voltages used to represent a 1 and a 0 are called *logic levels.* Ideally, one voltage level represents a HIGH and another voltage level represents a LOW. In a practical digital circuit, however, a HIGH can be any voltage between a specified minimum value and a specified maximum value. Likewise, a LOW can be any voltage between a specified minimum and a specified maximum. There can be no overlap between the accepted range of HIGH levels and the accepted range of LOW levels.

Figure 1–6 illustrates the general range of LOWs and HIGHs for a digital circuit. The variable $V_{H(max)}$ represents the maximum HIGH voltage value, and $V_{H(min)}$ represents the minimum HIGH voltage value. The maximum LOW voltage value is represented by $V_{L(max)}$, and the minimum LOW voltage value is represented by $V_{L(min)}$. The voltage values between $V_{L(max)}$ and $V_{H(min)}$ are unacceptable for proper operation. A voltage in the unacceptable range can appear as either a HIGH or a LOW to a given circuit. For example, the HIGH input values for a certain type of digital circuit technology called CMOS may range from 2 V to 3.3 V and the LOW input values may range from 0 V to 0.8 V. If a voltage of 2.5 V is applied, the circuit will accept it as a HIGH or binary 1. If a voltage of 0.5 V is applied, the circuit will accept it as a LOW or binary 0. For this type of circuit, voltages between 0.8 V and 2 V are unacceptable.

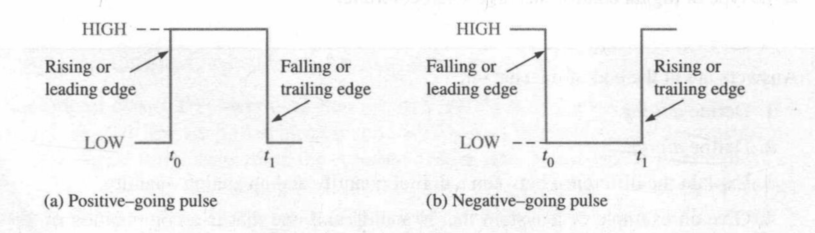


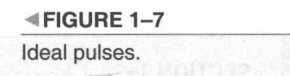
▲ **FIGURE 1–6**

Logic level ranges of voltage for a digital circuit.

Digital Waveforms

Digital waveforms consist of voltage levels that are changing back and forth between the HIGH and LOW levels or states. Figure 1–7(a) shows that a single positive-going **pulse** is generated when the voltage (or current) goes from its normally LOW level to its HIGH level and then back to its LOW level. The negative-going pulse in Figure 1–7(b) is generated when the voltage goes from its normally HIGH level to its LOW level and back to its HIGH level. A digital waveform is made up of a series of pulses.





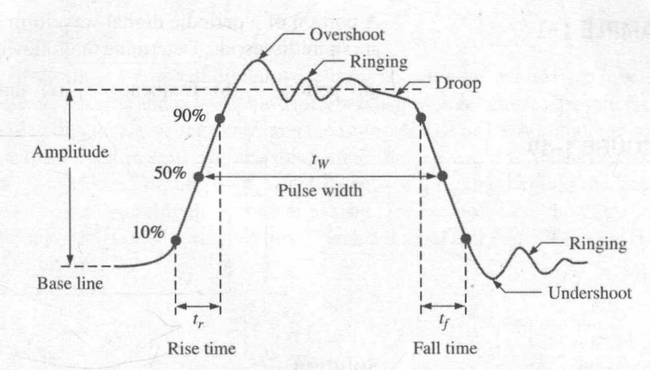
◀ **FIGURE 1–7**

Ideal pulses.

The Pulse

As indicated in Figure 1–7, a pulse has two edges: a **leading edge** that occurs first at time t_0 and a **trailing edge** that occurs last at time t_1. For a positive-going pulse, the leading edge is a rising edge, and the trailing edge is a falling edge. The pulses in Figure 1–7 are ideal because the rising and falling edges are assumed to change in zero time (instantaneously). In practice, these transitions never occur instantaneously, although for most digital work you can assume ideal pulses.

Figure 1–8 shows a nonideal pulse. In reality, all pulses exhibit some or all of these characteristics. The overshoot and ringing are sometimes produced by stray inductive and capacitive effects. The droop can be caused by stray capacitive and circuit resistance, forming an *RC* circuit with a low time constant.

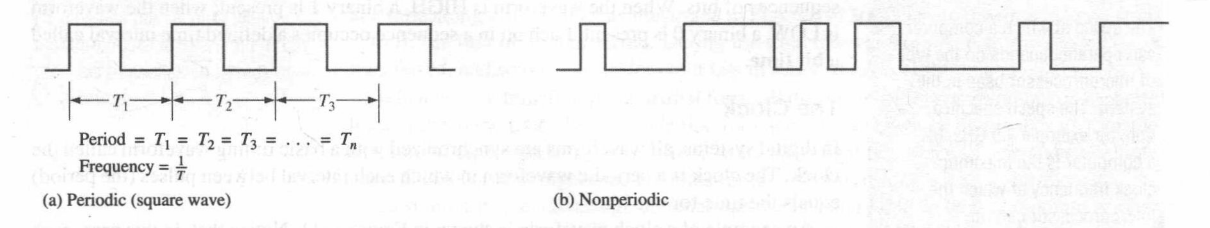


▶ **FIGURE 1–8**
Nonideal pulse characteristics.

The time required for a pulse to go from its LOW level to its HIGH level is called the **rise time (t_r)**, and the time required for the transition from the HIGH level to the LOW level is called the **fall time (t_f)**. In practice, it is common to measure rise time from 10% of the pulse **amplitude** (height from baseline) to 90% of the pulse amplitude and to measure the fall time from 90% to 10% of the pulse amplitude, as indicated in Figure 1–8. The bottom 10% and the top 10% of the pulse are not included in the rise and fall times because of the nonlinearities in the waveform in these areas. The **pulse width (t_W)** is a measure of the duration of the pulse and is often defined as the time interval between the 50% points on the rising and falling edges, as indicated in Figure 1–8.

Waveform Characteristics

Most waveforms encountered in digital systems are composed of series of pulses, sometimes called *pulse trains,* and can be classified as either periodic or nonperiodic. A **periodic** pulse waveform is one that repeats itself at a fixed interval, called a **period (T)**. The **frequency (f)** is the rate at which it repeats itself and is measured in hertz (Hz). A nonperiodic pulse waveform, of course, does not repeat itself at fixed intervals and may be composed of pulses of randomly differing pulse widths and/or randomly differing time intervals between the pulses. An example of each type is shown in Figure 1–9.



▲ **FIGURE 1–9**
Examples of digital waveforms.

The frequency (f) of a pulse (digital) waveform is the reciprocal of the period. The relationship between frequency and period is expressed as follows:

$$f = \frac{1}{T}$$ Equation 1–1

$$T = \frac{1}{f}$$ Equation 1–2

An important characteristic of a periodic digital waveform is its **duty cycle**, which is the ratio of the pulse width (t_W) to the period (T). It can be expressed as a percentage.

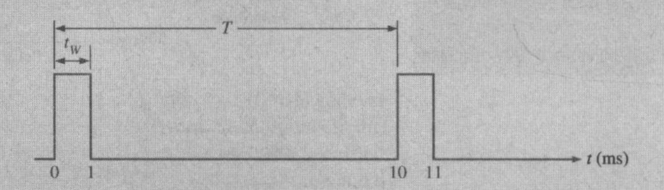
$$\textbf{Duty cycle} = \left(\frac{t_W}{T}\right)100\%$$ Equation 1–3

EXAMPLE 1–1

A portion of a periodic digital waveform is shown in Figure 1–10. The measurements are in milliseconds. Determine the following:

(a) period (b) frequency (c) duty cycle

▶ **FIGURE 1–10**



Solution

(a) The period (T) is measured from the edge of one pulse to the corresponding edge of the next pulse. In this case T is measured from leading edge to leading edge, as indicated. T equals **10 ms.**

(b) $f = \frac{1}{T} = \frac{1}{10 \text{ ms}} = \mathbf{100 \text{ Hz}}$

(c) Duty cycle $= \left(\frac{t_W}{T}\right)100\% = \left(\frac{1 \text{ ms}}{10 \text{ ms}}\right)100\% = \mathbf{10\%}$

Related Problem*

A periodic digital waveform has a pulse width of 25 μs and a period of 150 μs. Determine the frequency and the duty cycle.

*Answers are at the end of the chapter.

A Digital Waveform Carries Binary Information

InfoNote

The speed at which a computer can operate depends on the type of microprocessor used in the system. The speed specification, for example 3.5 GHz, of a computer is the maximum clock frequency at which the microprocessor can run.

Binary information that is handled by digital systems appears as waveforms that represent sequences of bits. When the waveform is HIGH, a binary 1 is present; when the waveform is LOW, a binary 0 is present. Each bit in a sequence occupies a defined time interval called a **bit time**.

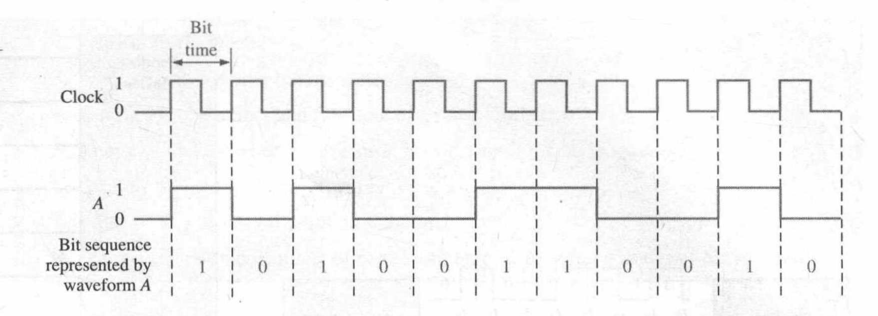
The Clock

In digital systems, all waveforms are synchronized with a basic timing waveform called the **clock.** The clock is a periodic waveform in which each interval between pulses (the period) equals the time for one bit.

An example of a clock waveform is shown in Figure 1–11. Notice that, in this case, each change in level of waveform A occurs at the leading edge of the clock waveform. In other cases, level changes occur at the trailing edge of the clock. During each bit time of the clock, waveform A is either HIGH or LOW. These HIGHs and LOWs represent a sequence of bits as indicated. A group of several bits can contain binary information, such as a number or a letter. The clock waveform itself does not carry information.

▶ **FIGURE 1–11**

Example of a clock waveform synchronized with a waveform representation of a sequence of bits.

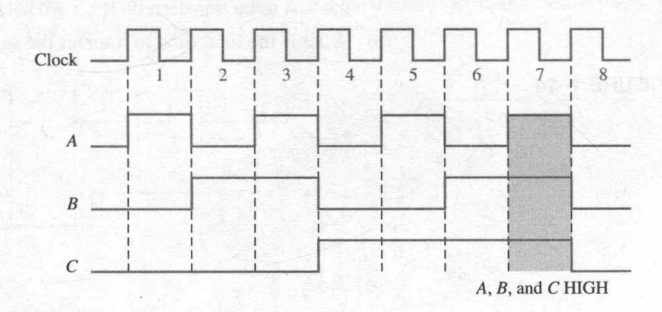


Timing Diagrams

A **timing diagram** is a graph of digital waveforms showing the actual time relationship of two or more waveforms and how each waveform changes in relation to the others. By looking at a timing diagram, you can determine the states (HIGH or LOW) of all the waveforms at any specified point in time and the exact time that a waveform changes state relative to the other waveforms. Figure 1–12 is an example of a timing diagram made up of four waveforms. From this timing diagram you can see, for example, that the three waveforms *A*, *B*, and *C* are HIGH only during bit time 7 (shaded area) and they all change back LOW at the end of bit time 7.

▶ **FIGURE 1–12**

Example of a timing diagram.



> **InfoNote**
>
> Universal Serial Bus (USB) is a serial bus standard for device interfacing. It was originally developed for the personal computer but has become widely used on many types of handheld and mobile devices. USB is expected to replace other serial and parallel ports. USB operated at 12 Mbps (million bits per second) when first introduced in 1995, but it now provides transmission speeds of up to 5 Gbps.

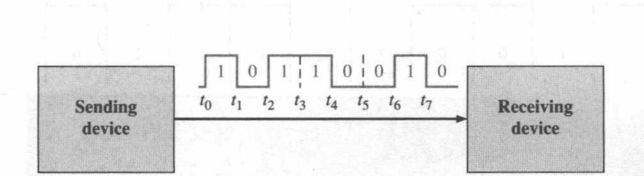
Data Transfer

Data refers to groups of bits that convey some type of information. Binary data, which are represented by digital waveforms, must be transferred from one device to another within a digital system or from one system to another in order to accomplish a given purpose. For example, numbers stored in binary form in the memory of a computer must be transferred to the computer's central processing unit in order to be added. The sum of the addition must then be transferred to a monitor for display and/or transferred back to the memory. As illustrated in Figure 1–13, binary data are transferred in two ways: serial and parallel.

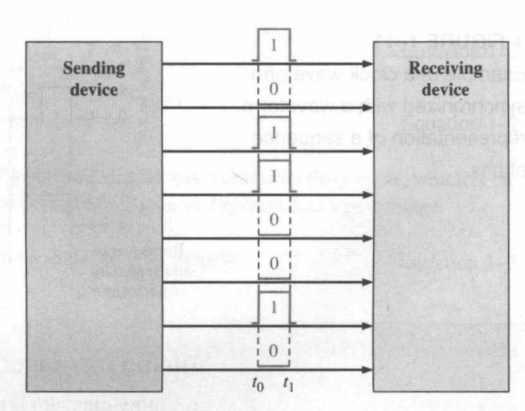
When bits are transferred in **serial** form from one point to another, they are sent one bit at a time along a single line, as illustrated in Figure 1–13(a). During the time interval from t_0 to t_1, the first bit is transferred. During the time interval from t_1 to t_2, the second bit is transferred, and so on. To transfer eight bits in series, it takes eight time intervals.

When bits are transferred in **parallel** form, all the bits in a group are sent out on separate lines at the same time. There is one line for each bit, as shown in Figure 1–13(b) for the example of eight bits being transferred. To transfer eight bits in parallel, it takes one time interval compared to eight time intervals for the serial transfer.

To summarize, an advantage of serial transfer of binary data is that a minimum of only one line is required. In parallel transfer, a number of lines equal to the number of bits to be transferred at one time is required. A disadvantage of serial transfer is that it takes longer to transfer a given number of bits than with parallel transfer at the same clock frequency. For example, if one bit can be transferred in 1 μs, then it takes 8 μs to serially transfer eight bits but only 1 μs to parallel transfer eight bits. A disadvantage of parallel transfer is that it takes more lines than serial transfer.



(a) Serial transfer of 8 bits of binary data. Interval t_0 to t_1 is first.



(b) Parallel transfer of 8 bits of binary data. The beginning time is t_0.

▶ **FIGURE 1–13**

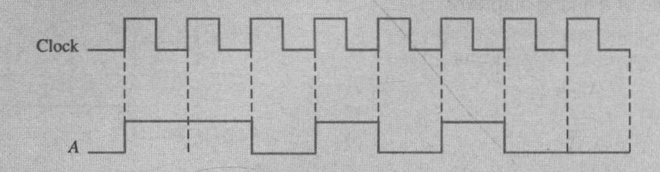
Illustration of serial and parallel transfer of binary data. Only the data lines are shown.

EXAMPLE 1-2

(a) Determine the total time required to serially transfer the eight bits contained in waveform A of Figure 1–14, and indicate the sequence of bits. The left-most bit is the first to be transferred. The 1 MHz clock is used as reference.

(b) What is the total time to transfer the same eight bits in parallel?

▶ **FIGURE 1–14**



Solution

(a) Since the frequency of the clock is 1 MHz, the period is

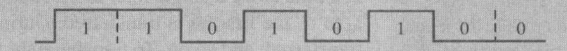
$$T = \frac{1}{f} = \frac{1}{1\text{ MHz}} = 1\ \mu\text{s}$$

It takes 1 μs to transfer each bit in the waveform. The total transfer time for 8 bits is

$$8 \times 1\ \mu\text{s} = \mathbf{8\ \mu s}$$

To determine the sequence of bits, examine the waveform in Figure 1–14 during each bit time. If waveform A is HIGH during the bit time, a 1 is transferred. If waveform A is LOW during the bit time, a 0 is transferred. The bit sequence is illustrated in Figure 1–15. The left-most bit is the first to be transferred.

▶ **FIGURE 1–15**



(b) A parallel transfer would take **1 μs** for all eight bits.

Related Problem

If binary data are transferred on a USB at the rate of 480 million bits per second (480 Mbps), how long will it take to serially transfer 16 bits?

SECTION 1-2 CHECKUP

2. What does *bit* mean?
3. What are the bits in a binary system?
4. How are the rise time and fall time of a pulse measured?
5. Knowing the period of a waveform, how do you find the frequency?
6. Explain what a clock waveform is.
7. What is the purpose of a timing diagram?
8. What is the main advantage of parallel transfer over serial transfer of binary data?

1-3 Fixed-Function Logic Devices

A monolithic **integrated circuit (IC)** is an electronic circuit that is constructed entirely on a single small chip of silicon. All the components that make up the circuit—transistors, diodes, resistors, and capacitors—are an integral part of that single chip. Fixed-function logic and programmable logic are two broad categories of digital ICs. In **fixed-function logic** devices, the logic functions are set by the manufacturer and cannot be altered.

Figure 1–16 shows a cutaway view of one type of fixed-function IC package with the circuit chip shown within the package. Points on the chip are connected to the package pins to allow input and output connections to the outside world.

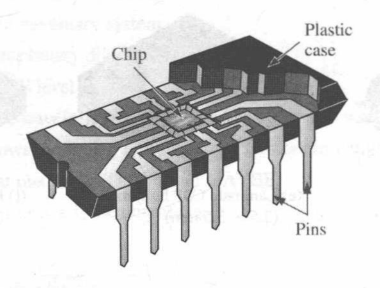
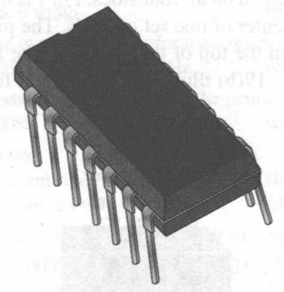


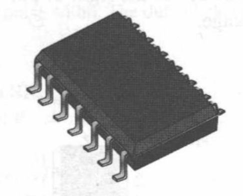
FIGURE 1–16 Cutaway view of one type of fixed-function IC package (dual in-line package) showing the chip mounted inside, with connections to input and output pins.

IC Packages

Integrated circuit (IC) packages are classified according to the way they are mounted on printed circuit boards (PCBs) as either through-hole mounted or surface mounted. The through-hole type packages have pins (leads) that are inserted through holes in the PCB and can be soldered to conductors on the opposite side. The most common type of through-hole package is the dual in-line package (**DIP**) shown in Figure 1–17(a).



(a) Dual in-line package (DIP)



(b) Small-outline IC (SOIC)

FIGURE 1–17 Examples of through-hole and surface-mounted devices. The DIP is larger than the SOIC with the same number of leads. This particular DIP is approximately 0.785 in. long, and the SOIC is approximately 0.385 in. long.

Another type of IC package uses surface-mount technology (**SMT**). Surface mounting is a space-saving alternative to through-hole mounting. The holes through the PCB are unnecessary for SMT. The pins of surface-mounted packages are soldered directly to conductors on one side of the board, leaving the other side free for additional circuits. Also, for

a circuit with the same number of pins, a surface-mounted package is much smaller than a dual in-line package because the pins are placed closer together. An example of a surface-mounted package is the small-outline integrated circuit (**SOIC**) shown in Figure 1–17(b).

Various types of SMT packages are available in a range of sizes, depending on the number of leads (more leads are required for more complex circuits and lead configurations). Examples of several types are shown in Figure 1–18. As you can see, the leads of the **SSOP** (shrink small-outline package) are formed into a "gull-wing" shape. The leads of the **PLCC** (plastic-leaded chip carrier) are turned under the package in a J-type shape. Instead of leads, the **LCC** (leadless ceramic chip) has metal contacts molded into its ceramic body. The LQFP (low-profile quad flat package) also has gull-wing leads. Both the CSP (chip scale package) and the FBGA (fine-pitch ball grid array) have contacts embedded in the bottom of the package.

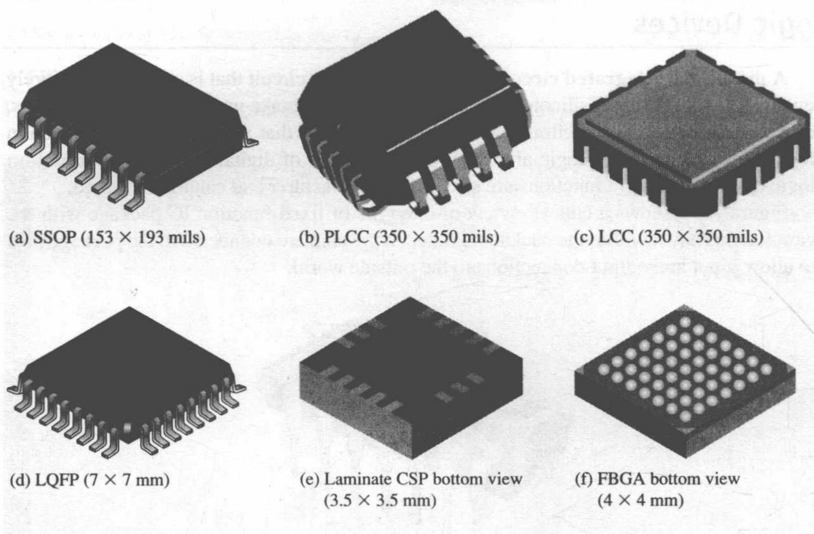


FIGURE 1–18
Examples of SMT package configurations. Parts (e) and (f) show bottom view.

Pin Numbering

All IC packages have a standard format for numbering the pins (leads). The dual in-line packages (DIPs) and the shrink small-outline packages (SSOP) have the numbering arrangement illustrated in Figure 1–19(a) for a 16-pin package. Looking at the top of the package, pin 1 is indicated by an identifier that can be either a small dot, a notch, or a beveled edge. The dot is always next to pin 1. Also, with the notch oriented upward, pin 1 is always the top left pin, as indicated. Starting with pin 1, the pin numbers increase as you go down, then across and up. The highest pin number is always to the right of the notch or opposite the dot.

The PLCC and LCC packages have leads arranged on all four sides. Pin 1 is indicated by a dot or other index mark and is located at the center of one set of leads. The pin numbers increase going counterclockwise as viewed from the top of the package. The highest pin number is always to the right of pin 1. Figure 1–19(b) illustrates this format for a 20-pin PLCC package.

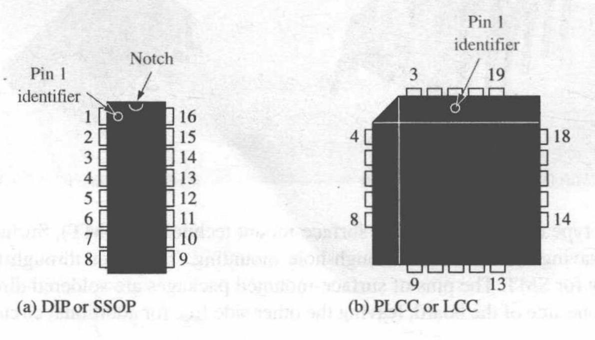


FIGURE 1–19
Pin numbering for two examples of standard types of IC packages. Top views are shown.

Complexity Classifications for Fixed-Function ICs

Fixed-function digital ICs are classified according to their complexity. They are listed here from the least complex to the most complex. The complexity figures stated here for SSI, MSI, LSI, VLSI, and ULSI are generally accepted, but definitions may vary from one source to another.

- **Small-scale integration (SSI)** describes fixed-function ICs that have up to ten equivalent gate circuits on a single chip, and they include basic gates and flip-flops.
- **Medium-scale integration (MSI)** describes integrated circuits that have from 10 to 100 equivalent gates on a chip. They include logic functions such as encoders, decoders, counters, registers, multiplexers, arithmetic circuits, small memories, and others.
- **Large-scale integration (LSI)** is a classification of ICs with complexities of from more than 100 to 10,000 equivalent gates per chip, including memories.
- **Very large-scale integration (VLSI)** describes integrated circuits with complexities of from more than 10,000 to 100,000 equivalent gates per chip.
- **Ultra large-scale integration (ULSI)** describes very large memories, larger **microprocessors**, and larger single-chip computers. Complexities of more than 100,000 equivalent gates per chip are classified as ULSI.

TRUE/FALSE QUIZ

Answers are at the end of the chapter.

1. An analog quantity is one having continuous values.
2. A digital quantity has ten discrete values.
3. There are two digits in the binary system.
4. The term *bit* is short for binary digit.
5. In positive logic, a LOW level represents a binary 1.
6. If the period of a pulse waveform increases, the frequency also increases.
7. A timing diagram shows the timing relationship of two or more digital waveforms.
8. The basic logic operations are AND, OR, and MAYBE.
9. If the input to an inverter is a 1, the output is a 0.

SELF-TEST

Answers are at the end of the chapter.

1. A quantity having continuous values is
   (a) a digital quantity **(b)** an analog quantity
   (c) a binary quantity **(d)** a natural quantity
2. The term *bit* means
   (a) a small amount of data **(b)** a 1 or a 0
   (c) binary digit **(d)** both answers (b) and (c)
3. The time interval on the leading edge of a pulse between 10% and 90% of the amplitude is the
   (a) rise time **(b)** fall time
   (c) pulse width **(d)** period
4. A pulse in a certain waveform occurs every 10 ms. The frequency is
   (a) 1 kHz **(b)** 1 Hz **(c)** 100 Hz **(d)** 10 Hz
5. In a certain digital waveform, the period is twice the pulse width. The duty cycle is
   (a) 100% **(b)** 200% **(c)** 50%
6. An inverter
   (a) performs the NOT operation **(b)** changes a HIGH to a LOW
   (c) changes a LOW to a HIGH **(d)** does all of the above
7. The output of an AND gate is HIGH when
   (a) any input is HIGH **(b)** all inputs are HIGH
   (c) no inputs are HIGH **(d)** both answers (a) and (b)
8. The output of an OR gate is HIGH when
   (a) any input is HIGH **(b)** all inputs are HIGH
   (c) no inputs are HIGH **(d)** both answers (a) and (b)
9. The device used to convert a binary number to a 7-segment display format is the
   (a) multiplexer **(b)** encoder
   (c) decoder **(d)** register

PROBLEMS

Answers to odd-numbered problems are at the end of the book.

Section 1–1 Digital and Analog Quantities

1. Name two advantages of digital data as compared to analog data.

2. Name an analog quantity other than temperature and sound.

3. List three common products that can have either a digital or analog output.

Section 1–2 Binary Digits, Logic Levels, and Digital Waveforms

4. Explain the difference between positive and negative logic.

5. Define the sequence of bits (1s and 0s) represented by each of the following sequences of levels:

(a) HIGH, HIGH, LOW, HIGH, LOW, LOW, LOW, HIGH
(b) LOW, LOW, LOW, HIGH, LOW, HIGH, LOW, HIGH, LOW

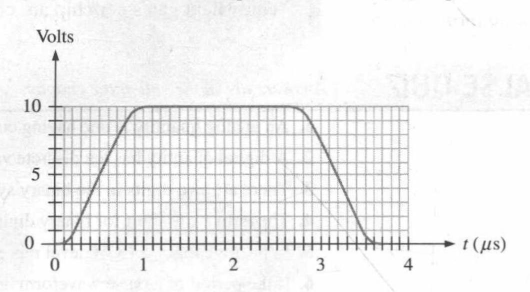
6. List the sequence of levels (HIGH and LOW) that represent each of the following bit sequences:

(a) 1 0 1 1 1 0 1 **(b)** 1 1 1 0 1 0 0 1

7. For the pulse shown in Figure 1–20, graphically determine the following:

(a) rise time **(b)** fall time **(c)** pulse width **(d)** amplitude

► FIGURE 1–20



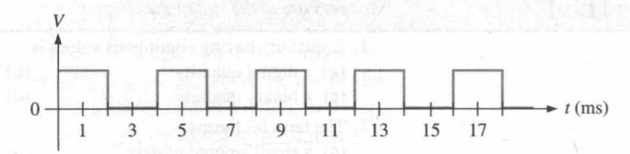
8. Determine the period of the digital waveform in Figure 1–21.

9. What is the frequency of the waveform in Figure 1–21?

10. Is the pulse waveform in Figure 1–21 periodic or nonperiodic?

11. Determine the duty cycle of the waveform in Figure 1–21.

► FIGURE 1–21

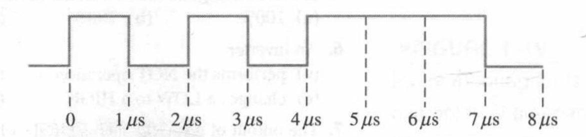


12. Determine the bit sequence represented by the waveform in Figure 1–22. A bit time is 1 μs in this case.

13. What is the total serial transfer time for the eight bits in Figure 1–22? What is the total parallel transfer time?

14. What is the period if the clock frequency is 3.5 GHz?

► FIGURE 1–22



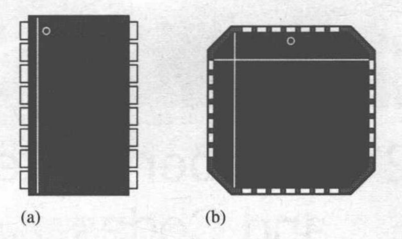
Section 1–3 Fixed-Function Logic Devices

15. A fixed-function digital IC chip has a complexity of 200 equivalent gates. How is it classified?

16. Explain the main difference between the DIP and SMT packages.

17. Label the pin numbers on the packages in Figure 1–23. Top views are shown.

▶ FIGURE 1–23



ANSWERS

SECTION CHECKUPS

Section 1–1 Digital and Analog Quantities

1. *Analog* means continuous.
2. *Digital* means discrete.
3. A digital quantity has a discrete set of values and an analog quantity has continuous values.
4. A public address system is analog. A CD player is analog and digital. A computer is all digital.
5. A mechatronic system consists of both mechanical and electronic components.

Section 1–2 Binary Digits, Logic Levels, and Digital Waveforms

1. Binary means having two states or values.
2. A bit is a binary digit.
3. The bits are 1 and 0.
4. Rise time: from 10% to 90% of amplitude. Fall time: from 90% to 10% of amplitude.
5. Frequency is the reciprocal of the period.
6. A clock waveform is a basic timing waveform from which other waveforms are derived.
7. A timing diagram shows the time relationship of two or more waveforms.
8. Parallel transfer is faster than serial transfer.

RELATED PROBLEMS FOR EXAMPLES

1–1 $f = 6.67$ kHz; Duty cycle $= 16.7\%$

1–2 Serial transfer: 3.33 ns

TRUE/FALSE QUIZ

1. T **2.** F **3.** T **4.** T **5.** F **6.** F **7.** T **8.** F **9.** T

SELF-TEST

1. (b) **2.** (d) **3.** (a) **4.** (c) **5.** (c) **6.** (d) **7.** (b) **8.** (d) **9.** (c)

Chapter 2 Number Systems, Operations, and Codes

CHAPTER OUTLINE

2–1 Decimal Numbers

In the **decimal** number system each of the ten digits, 0 through 9, represents a certain quantity. As you know, the ten symbols (**digits**) do not limit you to expressing only ten different quantities because you use the various digits in appropriate positions within a number to indicate the magnitude of the quantity. You can express quantities up through nine before running out of digits; if you wish to express a quantity greater than nine, you use two or more digits, and the position of each digit within the number tells you the magnitude it represents. If, for example, you wish to express the quantity twenty-three, you use (by their respective positions in the number) the digit 2 to represent the quantity twenty and the digit 3 to represent the quantity three, as illustrated below.

The decimal number system has ten digits.

The digit 2 has a weight of 10 in this position. → 2

The digit 3 has a weight of 1 in this position. → 3

$$2 \times 10 + 3 \times 1$$

$$20 + 3$$

$$23$$

The position of each digit in a decimal number indicates the magnitude of the quantity represented and can be assigned a **weight**. The weights for whole numbers are positive powers of ten that increase from right to left, beginning with $10^0 = 1$.

The decimal number system has a base of 10.

$$\ldots 10^5\ 10^4\ 10^3\ 10^2\ 10^1\ 10^0$$

For fractional numbers, the weights are negative powers of ten that decrease from left to right beginning with 10^{-1}.

$$10^2\ 10^1\ 10^0.10^{-1}\ 10^{-2}\ 10^{-3}\ \ldots$$

↑ Decimal point

The value of a digit is determined by its position in the number.

The value of a decimal number is the sum of the digits after each digit has been multiplied by its weight, as Examples 2–1 and 2–2 illustrate.

EXAMPLE 2-1

Express the decimal number 47 as a sum of the values of each digit.

Solution

The digit 4 has a weight of 10, which is 10^1, as indicated by its position. The digit 7 has a weight of 1, which is 10^0, as indicated by its position.

$$\begin{aligned} 47 &= (4 \times 10^1) + (7 \times 10^0) \\ &= (4 \times 10) + (7 \times 1) = \mathbf{40 + 7} \end{aligned}$$

Related Problem*

Determine the value of each digit in 939.

*Answers are at the end of the chapter.

EXAMPLE 2-2

Express the decimal number 568.23 as a sum of the values of each digit.

Solution

The whole number digit 5 has a weight of 100, which is 10^2, the digit 6 has a weight of 10, which is 10^1, the digit 8 has a weight of 1, which is 10^0, the fractional digit 2 has a weight of 0.1, which is 10^{-1}, and the fractional digit 3 has a weight of 0.01, which is 10^{-2}.

$$\begin{aligned} 568.23 &= (5 \times 10^2) + (6 \times 10^1) + (8 \times 10^0) + (2 \times 10^{-1}) + (3 \times 10^{-2}) \\ &= (5 \times 100) + (6 \times 10) + (8 \times 1) + (2 \times 0.1) + (3 \times 0.01) \\ &= \mathbf{500 + 60 + 8 + 0.2 + 0.03} \end{aligned}$$

Related Problem

Determine the value of each digit in 67.924.

SECTION 2-1 CHECKUP

Answers are at the end of the chapter.

1. What weight does the digit 7 have in each of the following numbers?

   (a) 1370 (b) 6725 (c) 7051 (d) 58.72

2. Express each of the following decimal numbers as a sum of the products obtained by multiplying each digit by its appropriate weight:

   (a) 51 (b) 137 (c) 1492 (d) 106.58

2–2 Binary Numbers

Counting in Binary

The binary number system has two digits (bits).

To learn to count in the binary system, first look at how you count in the decimal system. You start at zero and count up to nine before you run out of digits. You then start another digit position (to the left) and continue counting 10 through 99. At this point you have exhausted all two-digit combinations, so a third digit position is needed to count from 100 through 999.

A comparable situation occurs when you count in binary, except that you have only two digits, called *bits*. Begin counting: 0, 1. At this point you have used both digits, so include another digit position and continue: 10, 11. You have now exhausted all combinations of two digits, so a third position is required. With three digit positions you can continue to count: 100, 101, 110, and 111. Now you need a fourth digit position to continue, and so on. A binary count of zero through fifteen is shown in Table 2–1. Notice the patterns with which the 1s and 0s alternate in each column.

The binary number system has a base of 2.

TABLE 2–1

Decimal Number	Binary Number			
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

InfoNote

In processor operations, there are many cases where adding or subtracting 1 to a number stored in a counter is necessary. Processors have special instructions that use less time and generate less machine code than the ADD or SUB instructions. For the Intel processors, the INC (increment) instruction adds 1 to a number. For subtraction, the corresponding instruction is DEC (decrement), which subtracts 1 from a number.

As you have seen in Table 2–1, four bits are required to count from zero to 15. In general, with n bits you can count up to a number equal to $2^n - 1$.

$$\text{Largest decimal number} = 2^n - 1$$

For example, with five bits ($n = 5$) you can count from zero to thirty-one.

$$2^5 - 1 = 32 - 1 = 31$$

With six bits ($n = 6$) you can count from zero to sixty-three.

$$2^6 - 1 = 64 - 1 = 63$$

The Weighting Structure of Binary Numbers

A binary number is a weighted number. The right-most bit is the **LSB** (least significant bit) in a binary whole number and has a weight of $2^0 = 1$. The weights increase from right to left by a power of two for each bit. The left-most bit is the **MSB** (most significant bit); its weight depends on the size of the binary number.

The weight or value of a bit increases from right to left in a binary number.

Fractional numbers can also be represented in binary by placing bits to the right of the binary point, just as fractional decimal digits are placed to the right of the decimal point. The left-most bit is the MSB in a binary fractional number and has a weight of $2^{-1} = 0.5$. The fractional weights decrease from left to right by a negative power of two for each bit.

The weight structure of a binary number is

$$2^{n-1} \ldots 2^3\, 2^2\, 2^1\, 2^0\, .\, 2^{-1}\, 2^{-2} \ldots 2^{-n}$$

↑ Binary point

where n is the number of bits from the binary point. Thus, all the bits to the left of the binary point have weights that are positive powers of two, as previously discussed for whole numbers. All bits to the right of the binary point have weights that are negative powers of two, or fractional weights.

The powers of two and their equivalent decimal weights for an 8-bit binary whole number and a 6-bit binary fractional number are shown in Table 2–2. Notice that the weight

InfoNote

Processors use binary numbers to select memory locations. Each location is assigned a unique number called an *address*. Some microprocessors, for example, have 32 address lines which can select 2^{32} (4,294,967,296) unique locations.

doubles for each positive power of two and that the weight is halved for each negative power of two. You can easily extend the table by doubling the weight of the most significant positive power of two and halving the weight of the least significant negative power of two; for example, $2^9 = 512$ and $2^{-7} = 0.0078125$.

▼TABLE 2–2

Binary weights.

Positive Powers of Two (Whole Numbers)									Negative Powers of Two (Fractional Number)					
2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}
256	128	64	32	16	8	4	2	1	1/2	1/4	1/8	1/16	1/32	1/64
									0.5	0.25	0.125	0.625	0.03125	0.015625

Binary-to-Decimal Conversion

Add the weights of all 1s in a binary number to get the decimal value.

The decimal value of any binary number can be found by adding the weights of all bits that are 1 and discarding the weights of all bits that are 0.

EXAMPLE 2-3

Convert the binary whole number 1101101 to decimal.

Solution

Determine the weight of each bit that is a 1, and then find the sum of the weights to get the decimal number.

$$\text{Weight: } 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$$

$$\text{Binary number: } 1\ 1\ 0\ 1\ 1\ 0\ 1$$

$$1101101 = 2^6 + 2^5 + 2^3 + 2^2 + 2^0$$

$$= 64 + 32 + 8 + 4 + 1 = \mathbf{109}$$

Related Problem

Convert the binary number 10010001 to decimal.

EXAMPLE 2-4

Convert the fractional binary number 0.1011 to decimal.

Solution

Determine the weight of each bit that is a 1, and then sum the weights to get the decimal fraction.

$$\text{Weight: } 2^{-1}\ 2^{-2}\ 2^{-3}\ 2^{-4}$$

$$\text{Binary number: } 0.1\ 0\ 1\ 1$$

$$0.1011 = 2^{-1} + 2^{-3} + 2^{-4}$$

$$= 0.5 + 0.125 + 0.0625 = \mathbf{0.6875}$$

Related Problem

Convert the binary number 10.111 to decimal.

SECTION 2-2 CHECKUP

1. What is the largest decimal number that can be represented in binary with eight bits?
2. Determine the weight of the 1 in the binary number 10000.
3. Convert the binary number 10111101.011 to decimal.

2–3 Decimal-to-Binary Conversion

Sum-of-Weights Method

One way to find the binary number that is equivalent to a given decimal number is to determine the set of binary weights whose sum is equal to the decimal number. An easy way to remember binary weights is that the lowest is 1, which is 2^0, and that by doubling any weight, you get the next higher weight; thus, a list of seven binary weights would be 64, 32, 16, 8, 4, 2, 1 as you learned in the last section. The decimal number 9, for example, can be expressed as the sum of binary weights as follows:

To get the binary number for a given decimal number, find the binary weights that add up to the decimal number.

$$9 = 8 + 1 \quad \text{or} \quad 9 = 2^3 + 2^0$$

Placing 1s in the appropriate weight positions, 2^3 and 2^0, and 0s in the 2^2 and 2^1 positions determines the binary number for decimal 9.

2^3	2^2	2^1	2^0	
1	0	0	1	Binary number for decimal 9

EXAMPLE 2-5

Convert the following decimal numbers to binary:

(a) 12 **(b)** 25
(c) 58 **(d)** 82

Solution

(a) $12 = 8 + 4 = 2^3 + 2^2 \longrightarrow$ **1100**
(b) $25 = 16 + 8 + 1 = 2^4 + 2^3 + 2^0 \longrightarrow$ **11001**
(c) $58 = 32 + 16 + 8 + 2 = 2^5 + 2^4 + 2^3 + 2^1 \longrightarrow$ **111010**
(d) $82 = 64 + 16 + 2 = 2^6 + 2^4 + 2^1 \longrightarrow$ **1010010**

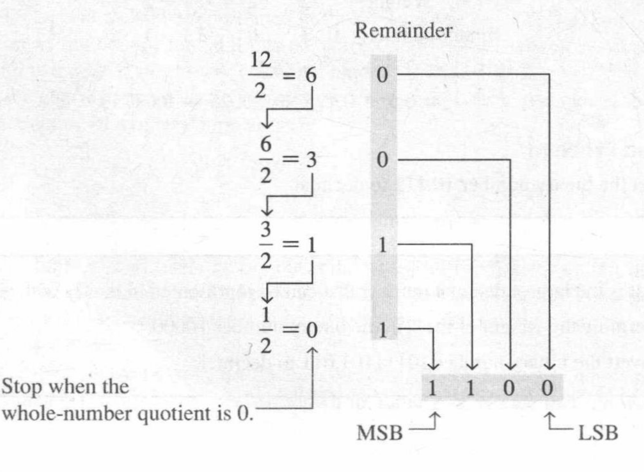
Related Problem

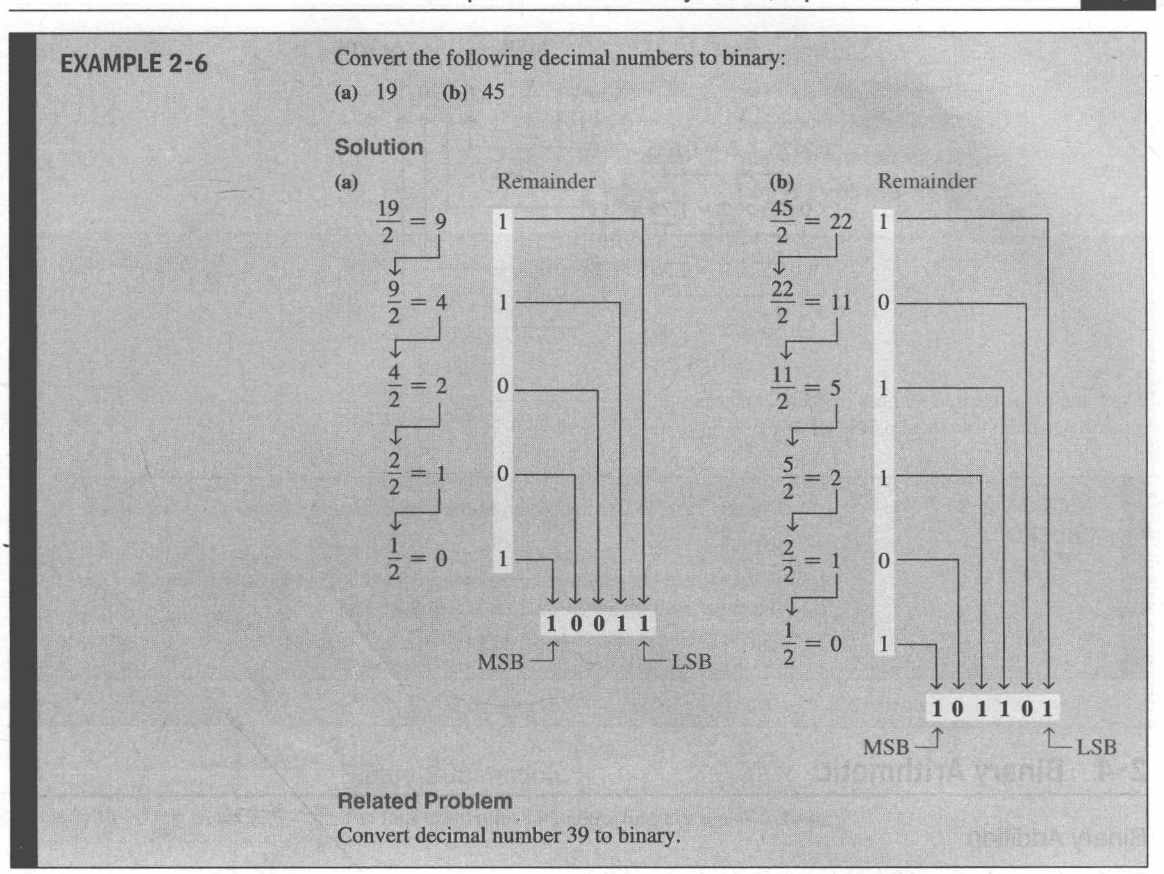
Convert the decimal number 125 to binary.

Repeated Division-by-2 Method

A systematic method of converting whole numbers from decimal to binary is the *repeated division-by-2* process. For example, to convert the decimal number 12 to binary, begin by dividing 12 by 2. Then divide each resulting quotient by 2 until there is a 0 whole-number quotient. The **remainders** generated by each division form the binary number. The first remainder to be produced is the LSB (least significant bit) in the binary number, and the last remainder to be produced is the MSB (most significant bit). This procedure is illustrated as follows for converting the decimal number 12 to binary.

To get the binary number for a given decimal number, divide the decimal number by 2 until the quotient is 0. Remainders form the binary number.





Converting Decimal Fractions to Binary

Examples 2–5 and 2–6 demonstrated whole-number conversions. Now let's look at fractional conversions. An easy way to remember fractional binary weights is that the most significant weight is 0.5, which is 2^{-1}, and that by halving any weight, you get the next lower weight; thus a list of four fractional binary weights would be 0.5, 0.25, 0.125, 0.0625.

Sum-of-Weights

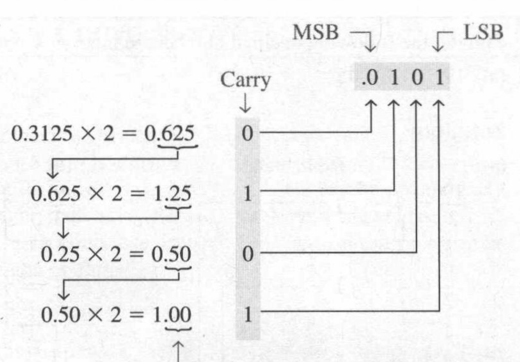
The sum-of-weights method can be applied to fractional decimal numbers, as shown in the following example:

$$0.625 = 0.5 + 0.125 = 2^{-1} + 2^{-3} = 0.101$$

There is a 1 in the 2^{-1} position, a 0 in the 2^{-2} position, and a 1 in the 2^{-3} position.

Repeated Multiplication by 2

As you have seen, decimal whole numbers can be converted to binary by repeated division by 2. Decimal fractions can be converted to binary by repeated multiplication by 2. For example, to convert the decimal fraction 0.3125 to binary, begin by multiplying 0.3125 by 2 and then multiplying each resulting fractional part of the product by 2 until the fractional product is zero or until the desired number of decimal places is reached. The carry digits, or **carries**, generated by the multiplications produce the binary number. The first carry produced is the MSB, and the last carry is the LSB. This procedure is illustrated as follows:



Continue to the desired number of decimal places or stop when the fractional part is all zeros.

SECTION 2-3 CHECKUP

1. Convert each decimal number to binary by using the sum-of-weights method:

   (a) 23 (b) 57 (c) 45.5

2. Convert each decimal number to binary by using the repeated division-by-2 method (repeated multiplication-by-2 for fractions):

   (a) 14 (b) 21 (c) 0.375

2-4 Binary Arithmetic

Binary Addition

The four basic rules for adding binary digits (bits) are as follows:

In binary 1 + 1 = 10, not 2.

0 + 0 = 0	Sum of 0 with a carry of 0
0 + 1 = 1	Sum of 1 with a carry of 0
1 + 0 = 1	Sum of 1 with a carry of 0
1 + 1 = 10	Sum of 0 with a carry of 1

Notice that the first three rules result in a single bit and in the fourth rule the addition of two 1s yields a binary two (10). When binary numbers are added, the last condition creates a sum of 0 in a given column and a carry of 1 over to the next column to the left, as illustrated in the following addition of 11 + 1:

```
Carry Carry
  1 ←┐ 1 ←┐
  0  │ 1  │ 1
+ 0  │ 0  │ 1
---------------
  1  └─0  └─0
```

In the right column, 1 + 1 = 0 with a carry of 1 to the next column to the left. In the middle column, 1 + 1 + 0 = 0 with a carry of 1 to the next column to the left. In the left column, 1 + 0 + 0 = 1.

When there is a carry of 1, you have a situation in which three bits are being added (a bit in each of the two numbers and a carry bit). This situation is illustrated as follows:

Carry bits

1	+ 0 + 0 = 01	Sum of 1 with a carry of 0
1	+ 1 + 0 = 10	Sum of 0 with a carry of 1
1	+ 0 + 1 = 10	Sum of 0 with a carry of 1
1	+ 1 + 1 = 11	Sum of 1 with a carry of 1

EXAMPLE 2-7

Add the following binary numbers:

(a) 11 + 11 **(b)** 100 + 10

(c) 111 + 11 **(d)** 110 + 100

Solution

The equivalent decimal addition is also shown for reference.

(a)
```
   11     3
 + 11   + 3
  110     6
```
(b)
```
  100     4
 + 10   + 2
  110     6
```
(c)
```
  111     7
 + 11   + 3
 1010    10
```
(d)
```
  110     6
+ 100   + 4
 1010    10
```

Related Problem

Add 1111 and 1100.

Binary Subtraction

In binary 10 − 1 = 1, not 9.

The four basic rules for subtracting bits are as follows:

$$0 - 0 = 0$$
$$1 - 1 = 0$$
$$1 - 0 = 1$$
$$10 - 1 = 1 \quad 0 - 1 \text{ with a borrow of } 1$$

When subtracting numbers, you sometimes have to borrow from the next column to the left. A borrow is required in binary only when you try to subtract a 1 from a 0. In this case, when a 1 is borrowed from the next column to the left, a 10 is created in the column being subtracted, and the last of the four basic rules just listed must be applied. Examples 2–8 and 2–9 illustrate binary subtraction; the equivalent decimal subtractions are also shown.

EXAMPLE 2-8

Perform the following binary subtractions:

(a) 11 − 01 **(b)** 11 − 10

Solution

(a)
```
  11     3
 −01    −1
  10     2
```
(b)
```
  11     3
 −10    −2
  01     1
```

No borrows were required in this example. The binary number 01 is the same as 1.

Related Problem

Subtract 100 from 111.

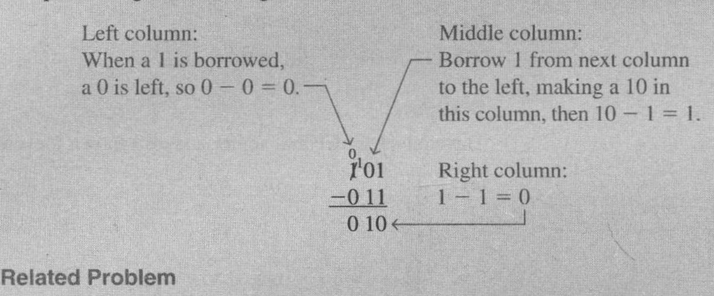
EXAMPLE 2-9

Subtract 011 from 101.

Solution

101	5
−011	−3
010	2

Let's examine exactly what was done to subtract the two binary numbers since a borrow is required. Begin with the right column.



Related Problem

Subtract 101 from 110.

Binary Multiplication

Binary multiplication of two bits is the same as multiplication of the decimal digits 0 and 1.

The four basic rules for multiplying bits are as follows:

$$0 \times 0 = 0$$
$$0 \times 1 = 0$$
$$1 \times 0 = 0$$
$$1 \times 1 = 1$$

Multiplication is performed with binary numbers in the same manner as with decimal numbers. It involves forming partial products, shifting each successive partial product left one place, and then adding all the partial products. Example 2–10 illustrates the procedure; the equivalent decimal multiplications are shown for reference.

EXAMPLE 2-10

Perform the following binary multiplications:

(a) 11 × 11 **(b)** 101 × 111

Solution

(a)

	Binary	Decimal
	11	3
	× 11	× 3
Partial products	11	9
	+11	
	1001	

(b)

	Binary	Decimal
	111	7
	× 101	× 5
Partial products	111	35
	000	
	+111	
	100011	

Related Problem

Multiply 1101 × 1010.

Binary Division

A calculator can be used to perform arithmetic operations with binary numbers as long as the capacity of the calculator is not exceeded.

Division in binary follows the same procedure as division in decimal, as Example 2–11 illustrates. The equivalent decimal divisions are also given.

EXAMPLE 2-11

Perform the following binary divisions:

(a) 110 ÷ 11 **(b)** 110 ÷ 10

Solution

```
         10      2              11      3
(a) 11)110    3)6      (b) 10)110    2)6
       11       6              10       6
       000      0              10       0
                               10
                               00
```

Related Problem

Divide 1100 by 100.

SECTION 2-4 CHECKUP

1. Perform the following binary additions:
   (a) 1101 + 1010 (b) 10111 + 01101
2. Perform the following binary subtractions:
   (a) 1101 − 0100 (b) 1001 − 0111
3. Perform the indicated binary operations:
   (a) 110 × 111 (b) 1100 ÷ 011

2-5 Complements of Binary Numbers

Finding the 1's Complement

Change each bit in a number to get the 1's complement.

The 1's **complement** of a binary number is found by changing all 1s to 0s and all 0s to 1s, as illustrated below:

```
1 0 1 1 0 0 1 0    Binary number
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
0 1 0 0 1 1 0 1    1's complement
```

The simplest way to obtain the 1's complement of a binary number with a digital circuit is to use parallel inverters (NOT circuits), as shown in Figure 2–1 for an 8-bit binary number.

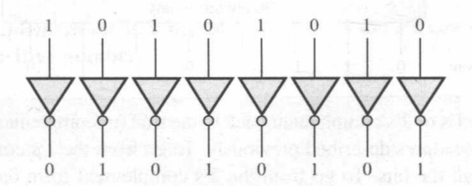


FIGURE 2–1
Example of inverters used to obtain the 1's complement of a binary number.

Finding the 2's Complement

Add 1 to the 1's complement to get the 2's complement.

The 2's complement of a binary number is found by adding 1 to the LSB of the 1's complement.

$$\text{2's complement} = (\text{1's complement}) + 1$$

EXAMPLE 2-12 Find the 2's complement of 10110010.

Solution

10110010	Binary number
01001101	1's complement
+ 1	Add 1
01001110	2's complement

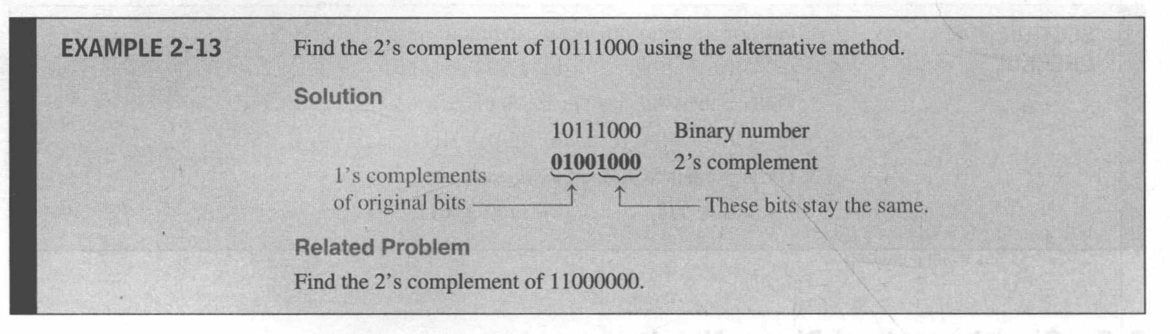
Related Problem

Determine the 2's complement of 11001011.

An alternative method of finding the 2's complement of a binary number is as follows:

Change all bits to the left of the least significant 1 to get 2's complement.

1. Start at the right with the LSB and write the bits as they are up to and including the first 1.
2. Take the 1's complements of the remaining bits.



EXAMPLE 2-13 Find the 2's complement of 10111000 using the alternative method.

Solution

10111000	Binary number
01001000	2's complement

1's complements of original bits → 01001; 000 ← These bits stay the same.

Related Problem

Find the 2's complement of 11000000.

The 2's complement of a negative binary number can be realized using inverters and an adder, as indicated in Figure 2–2. This illustrates how an 8-bit number can be converted to its 2's complement by first inverting each bit (taking the 1's complement) and then adding 1 to the 1's complement with an adder circuit.

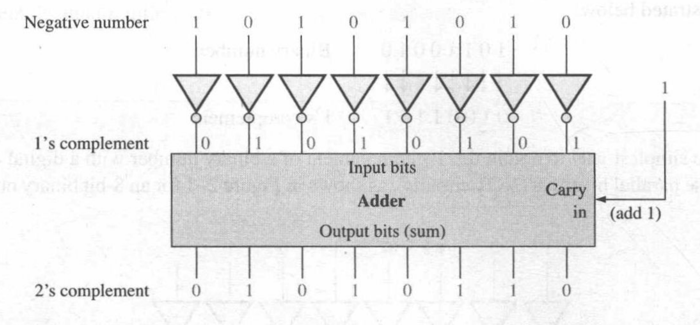


FIGURE 2–2
Example of obtaining the 2's complement of a negative binary number.

To convert from a 1's or 2's complement back to the true (uncomplemented) binary form, use the same two procedures described previously. To go from the 1's complement back to true binary, reverse all the bits. To go from the 2's complement form back to true binary, take the 1's complement of the 2's complement number and add 1 to the least significant bit.

SECTION 2-5 CHECKUP

1. Determine the 1's complement of each binary number:
   (a) 00011010 **(b)** 11110111 **(c)** 10001101
2. Determine the 2's complement of each binary number:
   (a) 00010110 **(b)** 11111100 **(c)** 10010001

2-6 Signed Numbers

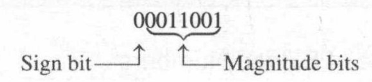
The Sign Bit

The left-most bit in a signed binary number is the **sign bit**, which tells you whether the number is positive or negative.

A 0 sign bit indicates a positive number, and a 1 sign bit indicates a negative number.

Sign-Magnitude Form

When a signed binary number is represented in sign-magnitude, the left-most bit is the sign bit and the remaining bits are the magnitude bits. The magnitude bits are in true (uncomplemented) binary for both positive and negative numbers. For example, the decimal number +25 is expressed as an 8-bit signed binary number using the sign-magnitude form as



The decimal number −25 is expressed as

10011001

Notice that the only difference between +25 and −25 is the sign bit because the magnitude bits are in true binary for both positive and negative numbers.

In the sign-magnitude form, a negative number has the same magnitude bits as the corresponding positive number but the sign bit is a 1 rather than a zero.

InfoNote

Processors use the 2's complement for negative integer numbers in arithmetic operations. The reason is that subtraction of a number is the same as adding the 2's complement of the number. Processors form the 2's complement by inverting the bits and adding 1, using special instructions that produce the same result as the adder in Figure 2-3.

1's Complement Form

Positive numbers in 1's complement form are represented the same way as the positive sign-magnitude numbers. Negative numbers, however, are the 1's complements of the corresponding positive numbers. For example, using eight bits, the decimal number −25 is expressed as the 1's complement of +25 (00011001) as

11100110

In the 1's complement form, a negative number is the 1's complement of the corresponding positive number.

2's Complement Form

Positive numbers in 2's complement form are represented the same way as in the sign-magnitude and 1's complement forms. Negative numbers are the 2's complements of the corresponding positive numbers. Again, using eight bits, let's take decimal number −25 and express it as the 2's complement of +25 (00011001). Inverting each bit and adding 1, you get

$$-25 = 11100111$$

In the 2's complement form, a negative number is the 2's complement of the corresponding positive number.

EXAMPLE 2-14

Express the decimal number −39 as an 8-bit number in the sign-magnitude, 1's complement, and 2's complement forms.

Solution

First, write the 8-bit number for +39.

00100111

In the *sign-magnitude form,* −39 is produced by changing the sign bit to a 1 and leaving the magnitude bits as they are. The number is

10100111

In the *1's complement form,* −39 is produced by taking the 1's complement of +39 (00100111).

11011000

In the *2's complement form,* −39 is produced by taking the 2's complement of +39 (00100111) as follows:

	11011000	1's complement
+	1	
	11011001	2's complement

Related Problem

Express +19 and −19 as 8-bit numbers in sign-magnitude, 1's complement, and 2's complement.

The Decimal Value of Signed Numbers

Sign-Magnitude

Decimal values of positive and negative numbers in the sign-magnitude form are determined by summing the weights in all the magnitude bit positions where there are 1s and ignoring those positions where there are zeros. The sign is determined by examination of the sign bit.

EXAMPLE 2-15

Determine the decimal value of this signed binary number expressed in sign-magnitude: 10010101.

Solution

The seven magnitude bits and their powers-of-two weights are as follows:

2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	1	0	1	0	1

Summing the weights where there are 1s,

$$16 + 4 + 1 = 21$$

The sign bit is 1; therefore, the decimal number is **−21**.

Related Problem

Determine the decimal value of the sign-magnitude number 01110111.

1's Complement

Decimal values of positive numbers in the 1's complement form are determined by summing the weights in all bit positions where there are 1s and ignoring those positions where there are zeros. Decimal values of negative numbers are determined by assigning a negative value to the weight of the sign bit, summing all the weights where there are 1s, and adding 1 to the result.

EXAMPLE 2-16

Determine the decimal values of the signed binary numbers expressed in 1's complement:

(a) 00010111 **(b)** 11101000

Solution

(a) The bits and their powers-of-two weights for the positive number are as follows:

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	0	1	0	1	1	1

Summing the weights where there are 1s,

$$16 + 4 + 2 + 1 = \mathbf{+23}$$

(b) The bits and their powers-of-two weights for the negative number are as follows. Notice that the negative sign bit has a weight of -2^7 or -128.

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	1	0	1	0	0	0

Summing the weights where there are 1s,

$$-128 + 64 + 32 + 8 = -24$$

Adding 1 to the result, the final decimal number is

$$-24 + 1 = \mathbf{-23}$$

Related Problem

Determine the decimal value of the 1's complement number 11101011.

2's Complement

Decimal values of positive and negative numbers in the 2's complement form are determined by summing the weights in all bit positions where there are 1s and ignoring those positions where there are zeros. The weight of the sign bit in a negative number is given a negative value.

EXAMPLE 2-17

Determine the decimal values of the signed binary numbers expressed in 2's complement:

(a) 01010110 (b) 10101010

Solution

(a) The bits and their powers-of-two weights for the positive number are as follows:

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	1	0	1	0	1	1	0

Summing the weights where there are 1s,

$$64 + 16 + 4 + 2 = \mathbf{+86}$$

(b) The bits and their powers-of-two weights for the negative number are as follows. Notice that the negative sign bit has a weight of $-2^7 = -128$.

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	1	0	1	0	1	0

Summing the weights where there are 1s,

$$-128 + 32 + 8 + 2 = \mathbf{-86}$$

Related Problem

Determine the decimal value of the 2's complement number 11010111.

From these examples, you can see why the 2's complement form is preferred for representing signed integer numbers: To convert to decimal, it simply requires a summation of weights regardless of whether the number is positive or negative. The 1's complement system requires adding 1 to the summation of weights for negative numbers but not for positive numbers. Also, the 1's complement form is generally not used because two representations of zero (00000000 or 11111111) are possible.

Range of Signed Integer Numbers

The range of magnitude values represented by binary numbers depends on the number of bits (n).

We have used 8-bit numbers for illustration because the 8-bit grouping is common in most computers and has been given the special name **byte**. With one byte or eight bits, you can represent 256 different numbers. With two bytes or sixteen bits, you can represent 65,536

different numbers. With four bytes or 32 bits, you can represent 4.295×10^9 different numbers. The formula for finding the number of different combinations of n bits is

$$\text{Total combinations} = 2^n$$

For 2's complement signed numbers, the range of values for n-bit numbers is

$$\text{Range} = -(2^{n-1}) \text{ to } +(2^{n-1} - 1)$$

where in each case there is one sign bit and $n - 1$ magnitude bits. For example, with four bits you can represent numbers in 2's complement ranging from $-(2^3) = -8$ to $2^3 - 1 = +7$. Similarly, with eight bits you can go from -128 to $+127$, with sixteen bits you can go from $-32,768$ to $+32,767$, and so on. There is one less positive number than there are negative numbers because zero is represented as a positive number (all zeros).

Floating-Point Numbers

To represent very large **integer** (whole) numbers, many bits are required. There is also a problem when numbers with both integer and fractional parts, such as 23.5618, need to be represented. The floating-point number system, based on scientific notation, is capable of representing very large and very small numbers without an increase in the number of bits and also for representing numbers that have both integer and fractional components.

A **floating-point number** (also known as a *real number*) consists of two parts plus a sign. The **mantissa** is the part of a floating-point number that represents the magnitude of the number and is between 0 and 1. The **exponent** is the part of a floating-point number that represents the number of places that the decimal point (or binary point) is to be moved.

A decimal example will be helpful in understanding the basic concept of floating-point numbers. Let's consider a decimal number which, in integer form, is 241,506,800. The mantissa is .2415068 and the exponent is 9. When the integer is expressed as a floating-point number, it is normalized by moving the decimal point to the left of all the digits so that the mantissa is a fractional number and the exponent is the power of ten. The floating-point number is written as

$$0.2415068 \times 10^9$$

For binary floating-point numbers, the format is defined by ANSI/IEEE Standard 754-1985 in three forms: *single-precision, double-precision,* and *extended-precision.* These all have the same basic formats except for the number of bits. Single-precision floating-point numbers have 32 bits, double-precision numbers have 64 bits, and extended-precision numbers have 80 bits. We will restrict our discussion to the single-precision floating-point format.

InfoNote

In addition to the CPU (central processing unit), computers use *coprocessors* to perform complicated mathematical calculations using floating-point numbers. The purpose is to increase performance by freeing up the CPU for other tasks. The mathematical coprocessor is also known as the floating-point unit (FPU).

Single-Precision Floating-Point Binary Numbers

In the standard format for a single-precision binary number, the sign bit (S) is the left-most bit, the exponent (E) includes the next eight bits, and the mantissa or fractional part (F) includes the remaining 23 bits, as shown next.

← 32 bits →

S	Exponent (E)	Mantissa (fraction, F)
1 bit	8 bits	23 bits

In the mantissa or fractional part, the binary point is understood to be to the left of the 23 bits. Effectively, there are 24 bits in the mantissa because in any binary number the left-most (most significant) bit is always a 1. Therefore, this 1 is understood to be there although it does not occupy an actual bit position.

The eight bits in the exponent represent a *biased exponent,* which is obtained by adding 127 to the actual exponent. The purpose of the bias is to allow very large or very small numbers without requiring a separate sign bit for the exponents. The biased exponent allows a range of actual exponent values from -126 to $+128$.

To illustrate how a binary number is expressed in floating-point format, let's use 1011010010001 as an example. First, it can be expressed as 1 plus a fractional binary number by moving the binary point 12 places to the left and then multiplying by the appropriate power of two.

$$1011010010001 = 1.011010010001 \times 2^{12}$$

Assuming that this is a positive number, the sign bit (S) is 0. The exponent, 12, is expressed as a biased exponent by adding it to 127 (12 + 127 = 139). The biased exponent (E) is expressed as the binary number 10001011. The mantissa is the fractional part (F) of the binary number, .011010010001. Because there is always a 1 to the left of the binary point in the power-of-two expression, it is not included in the mantissa. The complete floating-point number is

S	E	F
0	10001011	01101001000100000000000

Next, let's see how to evaluate a binary number that is already in floating-point format. The general approach to determining the value of a floating-point number is expressed by the following formula:

$$\text{Number} = (-1)^S(1 + F)(2^{E-127})$$

To illustrate, let's consider the following floating-point binary number:

S	E	F
1	10010001	10001110001000000000000

The sign bit is 1. The biased exponent is 10010001 = 145. Applying the formula, we get

$$\text{Number} = (-1)^1\,(1.10001110001)(2^{145-127})$$
$$= (-1)(1.10001110001)(2^{18}) = -1100011100010000000$$

This floating-point binary number is equivalent to −407,688 in decimal. Since the exponent can be any number between −126 and +128, extremely large and small numbers can be expressed. A 32-bit floating-point number can replace a binary integer number having 129 bits. Because the exponent determines the position of the binary point, numbers containing both integer and fractional parts can be represented.

There are two exceptions to the format for floating-point numbers: The number 0.0 is represented by all 0s, and infinity is represented by all 1s in the exponent and all 0s in the mantissa.

EXAMPLE 2-18

Convert the decimal number 3.248×10^4 to a single-precision floating-point binary number.

Solution

Convert the decimal number to binary.

$$3.248 \times 10^4 = 32480 = 111111011100000_2 = 1.11111011100000 \times 2^{14}$$

The MSB will not occupy a bit position because it is always a 1. Therefore, the mantissa is the fractional 23-bit binary number 11111011100000000000000 and the biased exponent is

$$14 + 127 = 141 = 10001101_2$$

The complete floating-point number is

0	**10001101**	**11111011100000000000000**

Related Problem

Determine the binary value of the following floating-point binary number:

0 10011000 10000100010100110000000

SECTION 2-6 CHECKUP

1. Express the decimal number +9 as an 8-bit binary number in the sign-magnitude system.
2. Express the decimal number −33 as an 8-bit binary number in the 1's complement system.
3. Express the decimal number −46 as an 8-bit binary number in the 2's complement system.
4. List the three parts of a signed, floating-point number.

2-7 Arithmetic Operations with Signed Numbers

Addition

The two numbers in an addition are the **addend** and the **augend**. The result is the **sum**. There are four cases that can occur when two signed binary numbers are added.

1. Both numbers positive
2. Positive number with magnitude larger than negative number
3. Negative number with magnitude larger than positive number
4. Both numbers negative

Let's take one case at a time using 8-bit signed numbers as examples. The equivalent decimal numbers are shown for reference.

Addition of two positive numbers yields a positive number.

Both numbers positive:

00000111   7
\+ 00000100   + 4
00001011   11

The sum is positive and is therefore in true (uncomplemented) binary.

Addition of a positive number and a smaller negative number yields a positive number.

Positive number with magnitude larger than negative number:

00001111   15
\+ 11111010   + −6
Discard carry ⟶ 1 00001001   9

The final carry bit is discarded. The sum is positive and therefore in true (uncomplemented) binary.

Addition of a positive number and a larger negative number or two negative numbers yields a negative number in 2's complement.

Negative number with magnitude larger than positive number:

00010000   16
\+ 11101000   + −24
11111000   −8

The sum is negative and therefore in 2's complement form.

Both numbers negative:

11111011   −5
\+ 11110111   + −9
Discard carry ⟶ 1 11110010   −14

The final carry bit is discarded. The sum is negative and therefore in 2's complement form.

In a computer, the negative numbers are stored in 2's complement form so, as you can see, the addition process is very simple: *Add the two numbers and discard any final carry bit.*

Overflow Condition

When two numbers are added and the number of bits required to represent the sum exceeds the number of bits in the two numbers, an **overflow** results as indicated by an incorrect sign bit. An overflow can occur only when both numbers are positive or both numbers are negative. If the sign bit of the result is different than the sign bit of the numbers that are added, overflow is indicated. The following 8-bit example will illustrate this condition.

01111101   125
\+ 00111010   + 58
10110111   183

Sign incorrect
Magnitude incorrect

In this example the sum of 183 requires eight magnitude bits. Since there are seven magnitude bits in the numbers (one bit is the sign), there is a carry into the sign bit which produces the overflow indication.

Numbers Added Two at a Time

Now let's look at the addition of a string of numbers, added two at a time. This can be accomplished by adding the first two numbers, then adding the third number to the sum of the first two, then adding the fourth number to this result, and so on. This is how computers add strings of numbers. The addition of numbers taken two at a time is illustrated in Example 2–19.

EXAMPLE 2-19

Add the signed numbers: 01000100, 00011011, 00001110, and 00010010.

Solution

The equivalent decimal additions are given for reference.

Decimal	Binary	
68	01000100	
+ 27	+ 00011011	Add 1st two numbers
95	01011111	1st sum
+ 14	+ 00001110	Add 3rd number
109	01101101	2nd sum
+ 18	+ 00010010	Add 4th number
127	**01111111**	Final sum

Related Problem

Add 00110011, 10111111, and 01100011. These are signed numbers.

Subtraction

Subtraction is addition with the sign of the subtrahend changed.

Subtraction is a special case of addition. For example, subtracting +6 (the **subtrahend**) from +9 (the **minuend**) is equivalent to adding −6 to +9. Basically, *the subtraction operation changes the sign of the subtrahend and adds it to the minuend.* The result of a subtraction is called the **difference**.

The sign of a positive or negative binary number is changed by taking its 2's complement.

For example, when you take the 2's complement of the positive number 00000100 (+4), you get 11111100, which is −4 as the following sum-of-weights evaluation shows:

$$-128 + 64 + 32 + 16 + 8 + 4 = -4$$

As another example, when you take the 2's complement of the negative number 11101101 (−19), you get 00010011, which is +19 as the following sum-of-weights evaluation shows:

$$16 + 2 + 1 = 19$$

Since subtraction is simply an addition with the sign of the subtrahend changed, the process is stated as follows:

When you subtract two binary numbers with the 2's complement method, it is important that both numbers have the same number of bits.

To subtract two signed numbers, take the 2's complement of the subtrahend and add. Discard any final carry bit.

Example 2–20 illustrates the subtraction process.

EXAMPLE 2-20

Perform each of the following subtractions of the signed numbers:

(a) 00001000 − 00000011 (b) 00001100 − 11110111

(c) 11100111 − 00010011 (d) 10001000 − 11100010

Solution

Like in other examples, the equivalent decimal subtractions are given for reference.

(a) In this case, $8 - 3 = 8 + (-3) = 5$.

	00001000	Minuend (+8)
	+ 11111101	2's complement of subtrahend (−3)
Discard carry ⟶	**1 00000101**	Difference (+5)

(b) In this case, 12 − (−9) = 12 + 9 = 21.

00001100	Minuend (+12)
+ 00001001	2's complement of subtrahend (+9)
00010101	Difference (+21)

(c) In this case, −25 − (+19) = −25 + (−19) = −44.

	11100111	Minuend (−25)
	+ 11101101	2's complement of subtrahend (−19)
Discard carry	**1 11010100**	Difference (−44)

(d) In this case, −120 − (−30) = −120 + 30 = −90.

10001000	Minuend (−120)
+ 00011110	2's complement of subtrahend (+30)
10100110	Difference (−90)

Related Problem

Subtract 01000111 from 01011000.

Multiplication

The numbers in a multiplication are the **multiplicand**, the **multiplier**, and the **product**. These are illustrated in the following decimal multiplication:

8	Multiplicand
× 3	Multiplier
24	Product

The multiplication operation in most computers is accomplished using addition. As you have already seen, subtraction is done with an adder; now let's see how multiplication is done.

Multiplication is equivalent to adding a number to itself a number of times equal to the multiplier.

Direct addition and *partial products* are two basic methods for performing multiplication using addition. In the direct addition method, you add the multiplicand a number of times equal to the multiplier. In the previous decimal example (8 × 3), three multiplicands are added: 8 + 8 + 8 = 24. The disadvantage of this approach is that it becomes very lengthy if the multiplier is a large number. For example, to multiply 350 × 75, you must add 350 to itself 75 times. Incidentally, this is why the term *times* is used to mean multiply.

When two binary numbers are multiplied, both numbers must be in true (uncomplemented) form. The direct addition method is illustrated in Example 2–21 adding two binary numbers at a time.

EXAMPLE 2-21

Multiply the signed binary numbers: 01001101 (multiplicand) and 00000100 (multiplier) using the direct addition method.

Solution

Since both numbers are positive, they are in true form, and the product will be positive. The decimal value of the multiplier is 4, so the multiplicand is added to itself four times as follows:

01001101	1st time
+ 01001101	2nd time
10011010	Partial sum
+ 01001101	3rd time
11100111	Partial sum
+ 01001101	4th time
100110100	Product

Since the sign bit of the multiplicand is 0, it has no effect on the outcome. All of the bits in the product are magnitude bits.

Related Problem

Multiply 01100001 by 00000110 using the direct addition method.

The partial products method is perhaps the more common one because it reflects the way you multiply longhand. The multiplicand is multiplied by each multiplier digit beginning with the least significant digit. The result of the multiplication of the multiplicand by a multiplier digit is called a *partial product.* Each successive partial product is moved (shifted) one place to the left and when all the partial products have been produced, they are added to get the final product. Here is a decimal example.

239	Multiplicand
× 123	Multiplier
717	1st partial product (3 × 239)
478	2nd partial product (2 × 239)
+ 239	3rd partial product (1 × 239)
29,397	Final product

The sign of the product of a multiplication depends on the signs of the multiplicand and the multiplier according to the following two rules:

- **If the signs are the same, the product is positive.**
- **If the signs are different, the product is negative.**

The basic steps in the partial products method of binary multiplication are as follows:

Step 1: Determine if the signs of the multiplicand and multiplier are the same or different. This determines what the sign of the product will be.

Step 2: Change any negative number to true (uncomplemented) form. Because most computers store negative numbers in 2's complement, a 2's complement operation is required to get the negative number into true form.

Step 3: Starting with the least significant multiplier bit, generate the partial products. When the multiplier bit is 1, the partial product is the same as the multiplicand. When the multiplier bit is 0, the partial product is zero. Shift each successive partial product one bit to the left.

Step 4: Add each successive partial product to the sum of the previous partial products to get the final product.

Step 5: If the sign bit that was determined in step 1 is negative, take the 2's complement of the product. If positive, leave the product in true form. Attach the sign bit to the product.

EXAMPLE 2-22

Multiply the signed binary numbers: 01010011 (multiplicand) and 11000101 (multiplier).

Solution

Step 1: The sign bit of the multiplicand is 0 and the sign bit of the multiplier is 1. The sign bit of the product will be 1 (negative).

Step 2: Take the 2's complement of the multiplier to put it in true form.

$$11000101 \longrightarrow 00111011$$

Step 3 and 4: The multiplication proceeds as follows. Notice that only the magnitude bits are used in these steps.

1010011	Multiplicand
× 0111011	Multiplier
1010011	1st partial product
+ 1010011	2nd partial product
11111001	Sum of 1st and 2nd
+ 0000000	3rd partial product
011111001	Sum
+ 1010011	4th partial product
1110010001	Sum
+ 1010011	5th partial product
10001100001	Sum
+ 1010011	6th partial product
1001100100001	Sum
+ 0000000	7th partial product
1001100100001	Final product

Step 5: Since the sign of the product is a 1 as determined in step 1, take the 2's complement of the product.

1001100100001 ⟶ 0110011011111

Attach the sign bit ⟶

1 0110011011111

Related Problem

Verify the multiplication is correct by converting to decimal numbers and performing the multiplication.

Division

The numbers in a division are the **dividend**, the **divisor**, and the **quotient**. These are illustrated in the following standard division format.

$$\frac{\text{dividend}}{\text{divisor}} = \text{quotient}$$

The division operation in computers is accomplished using subtraction. Since subtraction is done with an adder, division can also be accomplished with an adder.

The result of a division is called the *quotient;* the quotient is the number of times that the divisor will go into the dividend. This means that the divisor can be subtracted from the dividend a number of times equal to the quotient, as illustrated by dividing 21 by 7.

21	Dividend
− 7	1st subtraction of divisor
14	1st partial remainder
− 7	2nd subtraction of divisor
7	2nd partial remainder
− 7	3rd subtraction of divisor
0	Zero remainder

In this simple example, the divisor was subtracted from the dividend three times before a remainder of zero was obtained. Therefore, the quotient is 3.

The sign of the quotient depends on the signs of the dividend and the divisor according to the following two rules:

- **If the signs are the same, the quotient is positive.**
- **If the signs are different, the quotient is negative.**

When two binary numbers are divided, both numbers must be in true (uncomplemented) form. The basic steps in a division process are as follows:

Step 1: Determine if the signs of the dividend and divisor are the same or different. This determines what the sign of the quotient will be. Initialize the quotient to zero.

Step 2: Subtract the divisor from the dividend using 2's complement addition to get the first partial remainder and add 1 to the quotient. If this partial remainder is positive, go to step 3. If the partial remainder is zero or negative, the division is complete.

Step 3: Subtract the divisor from the partial remainder and add 1 to the quotient. If the result is positive, repeat for the next partial remainder. If the result is zero or negative, the division is complete.

Continue to subtract the divisor from the dividend and the partial remainders until there is a zero or a negative result. Count the number of times that the divisor is subtracted and you have the quotient. Example 2–23 illustrates these steps using 8-bit signed binary numbers.

EXAMPLE 2-23

Divide 01100100 by 00011001.

Solution

Step 1: The signs of both numbers are positive, so the quotient will be positive. The quotient is initially zero: 00000000.

Step 2: Subtract the divisor from the dividend using 2's complement addition (remember that final carries are discarded).

01100100	Dividend
+ 11100111	2's complement of divisor
01001011	Positive 1st partial remainder

Add 1 to quotient: 00000000 + 00000001 = 00000001.

Step 3: Subtract the divisor from the 1st partial remainder using 2's complement addition.

01001011	1st partial remainder
+ 11100111	2's complement of divisor
00110010	Positive 2nd partial remainder

Add 1 to quotient: 00000001 + 00000001 = 00000010.

Step 4: Subtract the divisor from the 2nd partial remainder using 2's complement addition.

00110010	2nd partial remainder
+ 11100111	2's complement of divisor
00011001	Positive 3rd partial remainder

Add 1 to quotient: 00000010 + 00000001 = 00000011.

Step 5: Subtract the divisor from the 3rd partial remainder using 2's complement addition.

00011001	3rd partial remainder
+ 11100111	2's complement of divisor
00000000	Zero remainder

Add 1 to quotient: 00000011 + 00000001 = **00000100** (final quotient). The process is complete.

Related Problem

Verify that the process is correct by converting to decimal numbers and performing the division.

SECTION 2-7 CHECKUP

1. List the four cases when numbers are added.
2. Add the signed numbers 00100001 and 10111100.
3. Subtract the signed numbers 00110010 from 01110111.
4. What is the sign of the product when two negative numbers are multiplied?
5. Multiply 01111111 by 00000101.
6. What is the sign of the quotient when a positive number is divided by a negative number?
7. Divide 00110000 by 00001100.

2-8 Hexadecimal Numbers

The **hexadecimal** number system has a base of sixteen; that is, it is composed of 16 **numeric** and alphabetic **characters**. Most digital systems process binary data in groups that are multiples of four bits, making the hexadecimal number very convenient because each hexadecimal digit represents a 4-bit binary number (as listed in Table 2–3).

The hexadecimal number system consists of digits 0–9 and letters A–F.

TABLE 2–3

Decimal	Binary	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Ten numeric digits and six alphabetic characters make up the hexadecimal number system. The use of letters A, B, C, D, E, and F to represent numbers may seem strange at first, but keep in mind that any number system is only a set of sequential symbols. If you understand what quantities these symbols represent, then the form of the symbols themselves is less important once you get accustomed to using them. We will use the subscript 16 to designate hexadecimal numbers to avoid confusion with decimal numbers. Sometimes you may see an "h" following a hexadecimal number.

InfoNote

With memories in the gigabyte (GB) range, specifying a memory address in binary is quite cumbersome. For example, it takes 32 bits to specify an address in a 4 GB memory. It is much easier to express a 32-bit code using 8 hexadecimal digits.

Counting in Hexadecimal

How do you count in hexadecimal once you get to F? Simply start over with another column and continue as follows:

. . ., E, F, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1A, 1B, 1C, 1D, 1E, 1F,
20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 2A, 2B, 2C, 2D, 2E, 2F, 30, 31, . . .

With two hexadecimal digits, you can count up to FF_{16}, which is decimal 255. To count beyond this, three hexadecimal digits are needed. For instance, 100_{16} is decimal 256, 101_{16} is decimal 257, and so forth. The maximum 3-digit hexadecimal number is FFF_{16}, or decimal 4095. The maximum 4-digit hexadecimal number is $FFFF_{16}$, which is decimal 65,535.

Binary-to-Hexadecimal Conversion

Converting a binary number to hexadecimal is a straightforward procedure. Simply break the binary number into 4-bit groups, starting at the right-most bit and replace each 4-bit group with the equivalent hexadecimal symbol.

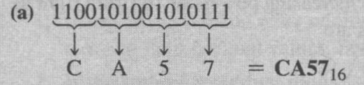
EXAMPLE 2-24

Convert the following binary numbers to hexadecimal:

(a) 1100101001010111 **(b)** 111111000101101001

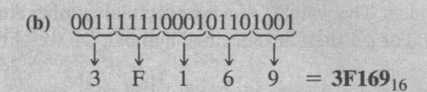
Solution

(a) 1100101001010111



C A 5 7 = $\mathbf{CA57_{16}}$

(b) 00111111000101101001



3 F 1 6 9 = $\mathbf{3F169_{16}}$

Two zeros have been added in part (b) to complete a 4-bit group at the left.

Related Problem

Convert the binary number 1001111011110011100 to hexadecimal.

Hexadecimal-to-Binary Conversion

Hexadecimal is a convenient way to represent binary numbers.

To convert from a hexadecimal number to a binary number, reverse the process and replace each hexadecimal symbol with the appropriate four bits.

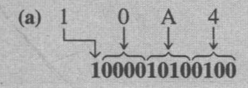
EXAMPLE 2-25

Determine the binary numbers for the following hexadecimal numbers:

(a) $10A4_{16}$ **(b)** $CF8E_{16}$ **(c)** 9742_{16}

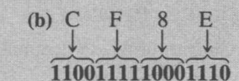
Solution

(a) 1 0 A 4



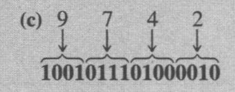
1000010100100

(b) C F 8 E



1100111110001110

(c) 9 7 4 2



1001011101000010

In part (a), the MSB is understood to have three zeros preceding it, thus forming a 4-bit group.

Related Problem

Convert the hexadecimal number 6BD3 to binary.

Conversion between hexadecimal and binary is direct and easy.

It should be clear that it is much easier to deal with a hexadecimal number than with the equivalent binary number. Since conversion is so easy, the hexadecimal system is widely used for representing binary numbers in programming, printouts, and displays.

Hexadecimal-to-Decimal Conversion

One way to find the decimal equivalent of a hexadecimal number is to first convert the hexadecimal number to binary and then convert from binary to decimal.

EXAMPLE 2-26

Convert the following hexadecimal numbers to decimal:

(a) $1C_{16}$ **(b)** $A85_{16}$

Solution

Remember, convert the hexadecimal number to binary first, then to decimal.

(a) 1 C

↓ ↓

$00011100 = 2^4 + 2^3 + 2^2 = 16 + 8 + 4 = \mathbf{28_{10}}$

(b) A 8 5 → $\underbrace{1010}\underbrace{1000}\underbrace{0101} = 2^{11} + 2^9 + 2^7 + 2^2 + 2^0 = 2048 + 512 + 128 + 4 + 1 = \mathbf{2693_{10}}$

Related Problem

Convert the hexadecimal number 6BD to decimal.

Another way to convert a hexadecimal number to its decimal equivalent is to multiply the decimal value of each hexadecimal digit by its weight and then take the sum of these products. The weights of a hexadecimal number are increasing powers of 16 (from right to left). For a 4-digit hexadecimal number, the weights are

16^3	16^2	16^1	16^0
4096	256	16	1

A calculator can be used to perform arithmetic operations with hexadecimal numbers.

EXAMPLE 2-27

Convert the following hexadecimal numbers to decimal:

(a) $E5_{16}$ **(b)** $B2F8_{16}$

Solution

Recall from Table 2–3 that letters A through F represent decimal numbers 10 through 15, respectively.

(a) $E5_{16} = (E \times 16) + (5 \times 1) = (14 \times 16) + (5 \times 1) = 224 + 5 = \mathbf{229_{10}}$

(b) $B2F8_{16} = (B \times 4096) + (2 \times 256) + (F \times 16) + (8 \times 1)$
$= (11 \times 4096) + (2 \times 256) + (15 \times 16) + (8 \times 1)$
$= 45{,}056 + 512 + 240 + 8 = \mathbf{45{,}816_{10}}$

Related Problem

Convert $60A_{16}$ to decimal.

Decimal-to-Hexadecimal Conversion

Repeated division of a decimal number by 16 will produce the equivalent hexadecimal number, formed by the remainders of the divisions. The first remainder produced is the least significant digit (LSD). Each successive division by 16 yields a remainder that becomes a digit in the equivalent hexadecimal number. This procedure is similar to repeated division by 2 for decimal-to-binary conversion that was covered in Section 2–3. Example 2–28 illustrates the procedure. Note that when a quotient has a fractional part, the fractional part is multiplied by the divisor to get the remainder.

EXAMPLE 2-28

Convert the decimal number 650 to hexadecimal by repeated division by 16.

Solution

Hexadecimal remainder

$\frac{650}{16} = 40.625 \rightarrow 0.625 \times 16 = 10 =$ A

$\frac{40}{16} = 2.5 \rightarrow 0.5 \times 16 = 8 =$ 8

$\frac{2}{16} = 0.125 \rightarrow 0.125 \times 16 = 2 =$ 2

Stop when whole number quotient is zero.

2 8 A Hexadecimal number

MSD LSD

Related Problem

Convert decimal 2591 to hexadecimal.

Hexadecimal Addition

Addition can be done directly with hexadecimal numbers by remembering that the hexadecimal digits 0 through 9 are equivalent to decimal digits 0 through 9 and that hexadecimal digits A through F are equivalent to decimal numbers 10 through 15. When adding two hexadecimal numbers, use the following rules. (Decimal numbers are indicated by a subscript 10.)

1. In any given column of an addition problem, think of the two hexadecimal digits in terms of their decimal values. For instance, $5_{16} = 5_{10}$ and $C_{16} = 12_{10}$.
2. If the sum of these two digits is 15_{10} or less, bring down the corresponding hexadecimal digit.
3. If the sum of these two digits is greater than 15_{10}, bring down the amount of the sum that exceeds 16_{10} and carry a 1 to the next column.

EXAMPLE 2-29

Add the following hexadecimal numbers:

(a) $23_{16} + 16_{16}$ **(b)** $58_{16} + 22_{16}$ **(c)** $2B_{16} + 84_{16}$ **(d)** $DF_{16} + AC_{16}$

Solution

(a) $23_{16} + 16_{16} = \mathbf{39_{16}}$

right column: $3_{16} + 6_{16} = 3_{10} + 6_{10} = 9_{10} = 9_{16}$
left column: $2_{16} + 1_{16} = 2_{10} + 1_{10} = 3_{10} = 3_{16}$

(b) $58_{16} + 22_{16} = \mathbf{7A_{16}}$

right column: $8_{16} + 2_{16} = 8_{10} + 2_{10} = 10_{10} = A_{16}$
left column: $5_{16} + 2_{16} = 5_{10} + 2_{10} = 7_{10} = 7_{16}$

(c) $2B_{16} + 84_{16} = \mathbf{AF_{16}}$

right column: $B_{16} + 4_{16} = 11_{10} + 4_{10} = 15_{10} = F_{16}$
left column: $2_{16} + 8_{16} = 2_{10} + 8_{10} = 10_{10} = A_{16}$

(d) $DF_{16} + AC_{16} = \mathbf{18B_{16}}$

right column: $F_{16} + C_{16} = 15_{10} + 12_{10} = 27_{10}$
$27_{10} - 16_{10} = 11_{10} = B_{16}$ with a 1 carry
left column: $D_{16} + A_{16} + 1_{16} = 13_{10} + 10_{10} + 1_{10} = 24_{10}$
$24_{10} - 16_{10} = 8_{10} = 8_{16}$ with a 1 carry

Related Problem

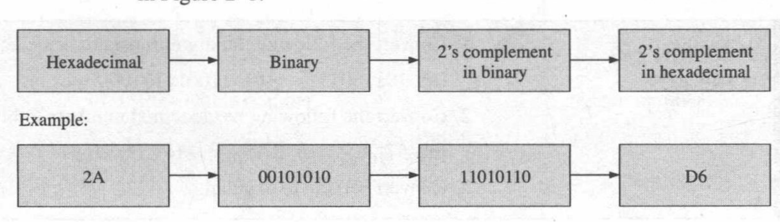
Add $4C_{16}$ and $3A_{16}$.

Hexadecimal Subtraction

As you have learned, the 2's complement allows you to subtract by adding binary numbers. Since a hexadecimal number can be used to represent a binary number, it can also be used to represent the 2's complement of a binary number.

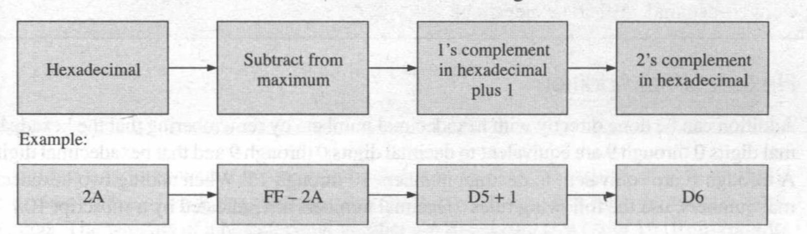
There are three ways to get the 2's complement of a hexadecimal number. Method 1 is the most common and easiest to use. Methods 2 and 3 are alternate methods.

Method 1: Convert the hexadecimal number to binary. Take the 2's complement of the binary number. Convert the result to hexadecimal. This is illustrated in Figure 2–3.



► **FIGURE 2-3**
Getting the 2's complement of a hexadecimal number, Method 1.

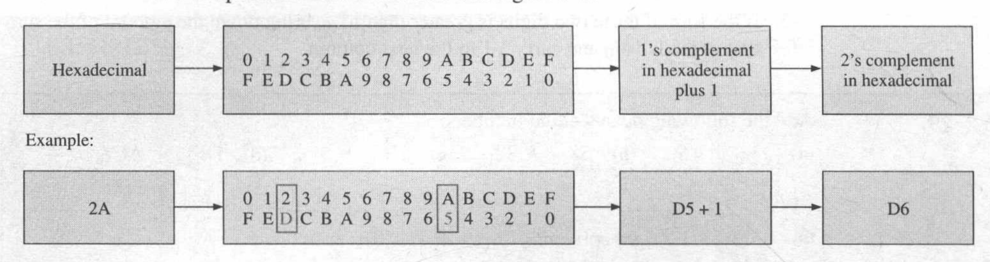
Method 2: Subtract the hexadecimal number from the maximum hexadecimal number and add 1. This is illustrated in Figure 2–4.



◄ **FIGURE 2-4**

Getting the 2's complement of a hexadecimal number, Method 2.

Method 3: Write the sequence of single hexadecimal digits. Write the sequence in reverse below the forward sequence. The 1's complement of each hex digit is the digit directly below it. Add 1 to the resulting number to get the 2's complement. This is illustrated in Figure 2–5.



▲ **FIGURE 2-5**

Getting the 2's complement of a hexadecimal number, Method 3.

EXAMPLE 2-30

Subtract the following hexadecimal numbers:

(a) $84_{16} - 2A_{16}$ **(b)** $C3_{16} - 0B_{16}$

Solution

(a) $2A_{16} = 00101010$

2's complement of $2A_{16} = 11010110 = D6_{16}$ (using Method 1)

$$\begin{array}{rl} 84_{16} & \\ +\ D6_{16} & \text{Add} \\ \hline \not{1}5A_{16} & \text{Drop carry, as in 2's complement addition} \end{array}$$

The difference is $\mathbf{5A_{16}}$.

(b) $0B_{16} = 00001011$

2's complement of $0B_{16} = 11110101 = F5_{16}$ (using Method 1)

$$\begin{array}{rl} C3_{16} & \\ +\ F5_{16} & \text{Add} \\ \hline \not{1}B8_{16} & \text{Drop carry} \end{array}$$

The difference is $\mathbf{B8_{16}}$.

Related Problem

Subtract 173_{16} from BCD_{16}.

SECTION 2-8 CHECKUP

1. Convert the following binary numbers to hexadecimal:
   (a) 10110011 **(b)** 110011101000
2. Convert the following hexadecimal numbers to binary:
   (a) 57_{16} **(b)** $3A5_{16}$ **(c)** $F80B_{16}$
3. Convert $9B30_{16}$ to decimal.

4. Convert the decimal number 573 to hexadecimal.
5. Add the following hexadecimal numbers directly:
   (a) $18_{16} + 34_{16}$ (b) $3F_{16} + 2A_{16}$
6. Subtract the following hexadecimal numbers:
   (a) $75_{16} - 21_{16}$ (b) $94_{16} - 5C_{16}$

2-9 Octal Numbers

The **octal** number system is composed of eight digits, which are

0, 1, 2, 3, 4, 5, 6, 7

To count above 7, begin another column and start over:

10, 11, 12, 13, 14, 15, 16, 17, 20, 21, . . .

The octal number system has a base of 8.

Counting in octal is similar to counting in decimal, except that the digits 8 and 9 are not used. To distinguish octal numbers from decimal numbers or hexadecimal numbers, we will use the subscript 8 to indicate an octal number. For instance, 15_8 in octal is equivalent to 13_{10} in decimal and D in hexadecimal. Sometimes you may see an "o" or a "Q" following an octal number.

Octal-to-Decimal Conversion

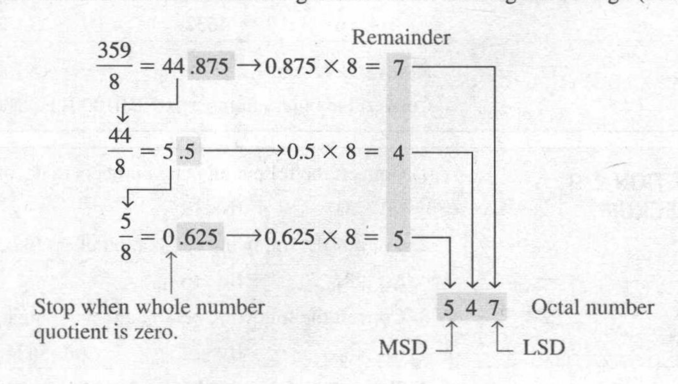
Since the octal number system has a base of eight, each successive digit position is an increasing power of eight, beginning in the right-most column with 8^0. The evaluation of an octal number in terms of its decimal equivalent is accomplished by multiplying each digit by its weight and summing the products, as illustrated here for 2374_8.

Weight: $8^3\ 8^2\ 8^1\ 8^0$
Octal number: 2 3 7 4

$$
\begin{aligned}
2374_8 &= (2 \times 8^3) + (3 \times 8^2) + (7 \times 8^1) + (4 \times 8^0) \\
&= (2 \times 512) + (3 \times 64) + (7 \times 8) + (4 \times 1) \\
&= 1024 + 192 + 56 + 4 = 1276_{10}
\end{aligned}
$$

Decimal-to-Octal Conversion

A method of converting a decimal number to an octal number is the repeated division-by-8 method, which is similar to the method used in the conversion of decimal numbers to binary or to hexadecimal. To show how it works, let's convert the decimal number 359 to octal. Each successive division by 8 yields a remainder that becomes a digit in the equivalent octal number. The first remainder generated is the least significant digit (LSD).



Octal-to-Binary Conversion

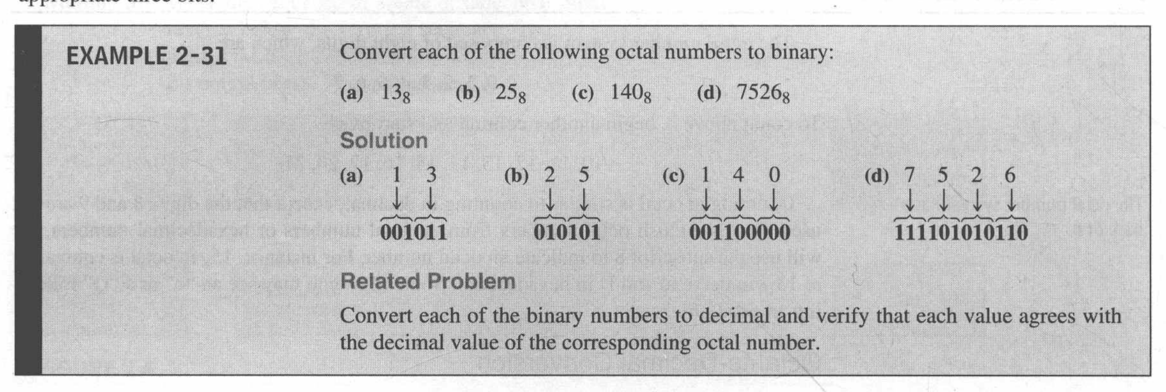
Because each octal digit can be represented by a 3-bit binary number, it is very easy to convert from octal to binary. Each octal digit is represented by three bits as shown in Table 2–4.

Octal is a convenient way to represent binary numbers, but it is not as commonly used as hexadecimal.

Octal Digit	0	1	2	3	4	5	6	7
Binary	000	001	010	011	100	101	110	111

◄ **TABLE 2–4**
Octal/binary conversion.

To convert an octal number to a binary number, simply replace each octal digit with the appropriate three bits.



EXAMPLE 2-31

Convert each of the following octal numbers to binary:

(a) 13_8 **(b)** 25_8 **(c)** 140_8 **(d)** 7526_8

Solution

(a) 1 3 → **001011**

(b) 2 5 → **010101**

(c) 1 4 0 → **001100000**

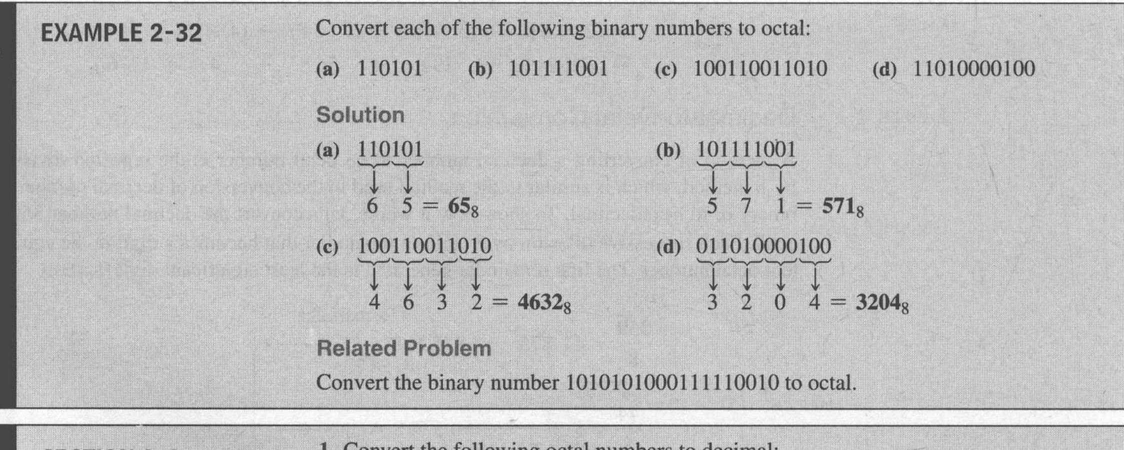
(d) 7 5 2 6 → **111101010110**

Related Problem

Convert each of the binary numbers to decimal and verify that each value agrees with the decimal value of the corresponding octal number.

Binary-to-Octal Conversion

Conversion of a binary number to an octal number is the reverse of the octal-to-binary conversion. The procedure is as follows: Start with the right-most group of three bits and, moving from right to left, convert each 3-bit group to the equivalent octal digit. If there are not three bits available for the left-most group, add either one or two zeros to make a complete group. These leading zeros do not affect the value of the binary number.



EXAMPLE 2-32

Convert each of the following binary numbers to octal:

(a) 110101 **(b)** 101111001 **(c)** 100110011010 **(d)** 11010000100

Solution

(a) 110101 → 6 5 = $\mathbf{65_8}$

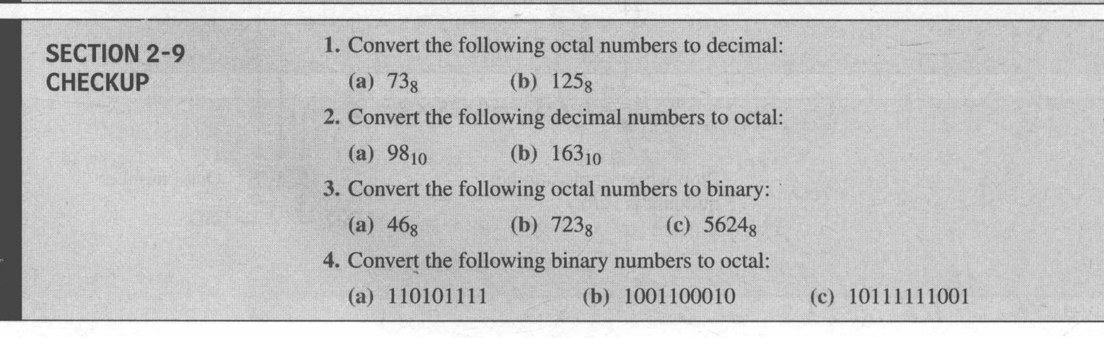
(b) 101111001 → 5 7 1 = $\mathbf{571_8}$

(c) 100110011010 → 4 6 3 2 = $\mathbf{4632_8}$

(d) 011010000100 → 3 2 0 4 = $\mathbf{3204_8}$

Related Problem

Convert the binary number 1010101000111110010 to octal.



SECTION 2-9 CHECKUP

1. Convert the following octal numbers to decimal:
   (a) 73_8 (b) 125_8
2. Convert the following decimal numbers to octal:
   (a) 98_{10} (b) 163_{10}
3. Convert the following octal numbers to binary:
   (a) 46_8 (b) 723_8 (c) 5624_8
4. Convert the following binary numbers to octal:
   (a) 110101111 (b) 1001100010 (c) 10111111001

2-10 Binary Coded Decimal (BCD)

In BCD, 4 bits represent each decimal digit.

The 8421 BCD Code

The 8421 code is a type of **BCD** (binary coded decimal) code. Binary coded decimal means that each decimal digit, 0 through 9, is represented by a binary code of four bits. The designation 8421 indicates the binary weights of the four bits ($2^3, 2^2, 2^1, 2^0$). The ease of conversion between 8421 code numbers and the familiar decimal numbers is the main advantage of this code. All you have to remember are the ten binary combinations that represent the ten decimal digits as shown in Table 2–5. The 8421 code is the predominant BCD code, and when we refer to BCD, we always mean the 8421 code unless otherwise stated.

▶ **TABLE 2–5**
Decimal/BCD conversion.

Decimal Digit	0	1	2	3	4	5	6	7	8	9
BCD	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

Invalid Codes

You should realize that, with four bits, sixteen numbers (0000 through 1111) can be represented but that, in the 8421 code, only ten of these are used. The six code combinations that are not used—1010, 1011, 1100, 1101, 1110, and 1111—are invalid in the 8421 BCD code.

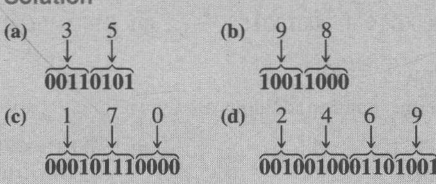
To express any decimal number in BCD, simply replace each decimal digit with the appropriate 4-bit code, as shown by Example 2–33.

EXAMPLE 2-33

Convert each of the following decimal numbers to BCD:

(a) 35 **(b)** 98 **(c)** 170 **(d)** 2469

Solution



Related Problem

Convert the decimal number 9673 to BCD.

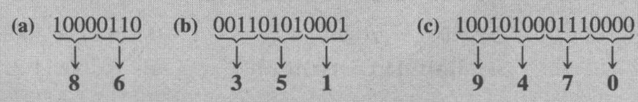
It is equally easy to determine a decimal number from a BCD number. Start at the right-most bit and break the code into groups of four bits. Then write the decimal digit represented by each 4-bit group.

EXAMPLE 2-34

Convert each of the following BCD codes to decimal:

(a) 10000110 **(b)** 001101010001 **(c)** 1001010001110000

Solution



Related Problem

Convert the BCD code 10000010001001110110 to decimal.

Applications

Digital clocks, digital thermometers, digital meters, and other devices with seven-segment displays typically use BCD code to simplify the displaying of decimal numbers. BCD is not as efficient as straight binary for calculations, but it is particularly useful if only limited processing is required, such as in a digital thermometer.

> **InfoNote**
>
> BCD is sometimes used for arithmetic operations in processors. To represent BCD numbers in a processor, they usually are "packed," so that eight bits have two BCD digits. Normally, a processor will add numbers as if they were straight binary. Special instructions are available for computer programmers to correct the results when BCD numbers are added or subtracted. For example, in Assembly Language, the programmer will include a DAA (Decimal Adjust for Addition) instruction to automatically correct the answer to BCD following an addition.

BCD Addition

BCD is a numerical code and can be used in arithmetic operations. Addition is the most important operation because the other three operations (subtraction, multiplication, and division) can be accomplished by the use of addition. Here is how to add two BCD numbers:

Step 1: Add the two BCD numbers, using the rules for binary addition in Section 2–4.

Step 2: If a 4-bit sum is equal to or less than 9, it is a valid BCD number.

Step 3: If a 4-bit sum is greater than 9, or if a carry out of the 4-bit group is generated, it is an invalid result. Add 6 (0110) to the 4-bit sum in order to skip the six invalid states and return the code to 8421. If a carry results when 6 is added, simply add the carry to the next 4-bit group.

Example 2–35 illustrates BCD additions in which the sum in each 4-bit column is equal to or less than 9, and the 4-bit sums are therefore valid BCD numbers. Example 2–36 illustrates the procedure in the case of invalid sums (greater than 9 or a carry).

An alternative method to add BCD numbers is to convert them to decimal, perform the addition, and then convert the answer back to BCD.

EXAMPLE 2-35

Add the following BCD numbers:

(a) 0011 + 0100 **(b)** 00100011 + 00010101

(c) 10000110 + 00010011 **(d)** 010001010000 + 010000010111

Solution

The decimal number additions are shown for comparison.

```
(a)   0011        3        (b)   0010 0011       23
    + 0100      + 4            + 0001 0101     + 15
      ----      ---              ---------     ----
      0111        7              0011 1000       38

(c)   1000 0110      86    (d)   0100 0101 0000      450
    + 0001 0011    + 13        + 0100 0001 0111    + 417
      ---------    ----          --------------    -----
      1001 1001      99          1000 0110 0111      867
```

Note that in each case the sum in any 4-bit column does not exceed 9, and the results are valid BCD numbers.

Related Problem

Add the BCD numbers: 1001000001000011 + 0000100100100101.

EXAMPLE 2-36

Add the following BCD numbers:

(a) 1001 + 0100 **(b)** 1001 + 1001

(c) 00010110 + 00010101 **(d)** 01100111 + 01010011

Solution

The decimal number additions are shown for comparison.

Solution

The decimal number additions are shown for comparison.

(a)

	1001		9
	+ 0100		+4
	1101	Invalid BCD number (>9)	13
	+ 0110	Add 6	
0001	**0011**	Valid BCD number	
1	3		

(b)

	1001		9
	+ 1001		+ 9
1	0010	Invalid because of carry	18
	+ 0110	Add 6	
0001	**1000**	Valid BCD number	
1	8		

(c)

0001	0110		16
+ 0001	0101		+ 15
0010	1011	Right group is invalid (>9), left group is valid.	31
	+ 0110	Add 6 to invalid code. Add carry, 0001, to next group.	
0011	**0001**	Valid BCD number	
3	1		

(d)

	0110	0111		67
	+ 0101	0011		+ 53
	1011	1010	Both groups are invalid (>9)	120
	+ 0110	+ 0110	Add 6 to both groups	
0001	**0010**	**0000**	Valid BCD number	
1	2	0		

Related Problem

Add the BCD numbers: 01001000 + 00110100.

SECTION 2-10 CHECKUP

1. What is the binary weight of each 1 in the following BCD numbers?
   (a) 0010 (b) 1000 (c) 0001 (d) 0100
2. Convert the following decimal numbers to BCD:
   (a) 6 (b) 15 (c) 273 (d) 849
3. What decimal numbers are represented by each BCD code?
   (a) 10001001 (b) 001001111000 (c) 000101010111
4. In BCD addition, when is a 4-bit sum invalid?

2-11 Digital Codes

The Gray Code

The **Gray code** is unweighted and is not an arithmetic code; that is, there are no specific weights assigned to the bit positions. The important feature of the Gray code is that *it exhibits only a single bit change from one code word to the next in sequence.* This property is important in many applications, such as shaft position encoders, where error susceptibility increases with the number of bit changes between adjacent numbers in a sequence.

The single bit change characteristic of the Gray code minimizes the chance for error.

Table 2–6 is a listing of the 4-bit Gray code for decimal numbers 0 through 15. Binary numbers are shown in the table for reference. Like binary numbers, *the Gray code can have any number of bits.* Notice the single-bit change between successive Gray code words. For instance, in going from decimal 3 to decimal 4, the Gray code changes from 0010 to 0110, while the binary code changes from 0011 to 0100, a change of three bits. The only bit change in the Gray code is in the third bit from the right: the other bits remain the same.

Decimal	Binary	Gray Code	Decimal	Binary	Gray Code
0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1101
2	0010	0011	10	1010	1111
3	0011	0010	11	1011	1110
4	0100	0110	12	1100	1010
5	0101	0111	13	1101	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000

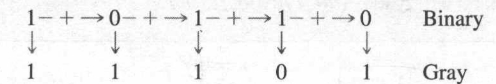
TABLE 2–6
Four-bit Gray code.

Binary-to-Gray Code Conversion

Conversion between binary code and Gray code is sometimes useful. The following rules explain how to convert from a binary number to a Gray code word:

1. The most significant bit (left-most) in the Gray code is the same as the corresponding MSB in the binary number.
2. Going from left to right, add each adjacent pair of binary code bits to get the next Gray code bit. Discard carries.

For example, the conversion of the binary number 10110 to Gray code is as follows:



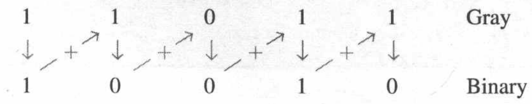
The Gray code is 11101.

Gray-to-Binary Code Conversion

To convert from Gray code to binary, use a similar method; however, there are some differences. The following rules apply:

1. The most significant bit (left-most) in the binary code is the same as the corresponding bit in the Gray code.
2. Add each binary code bit generated to the Gray code bit in the next adjacent position. Discard carries.

For example, the conversion of the Gray code word 11011 to binary is as follows:



The binary number is 10010.

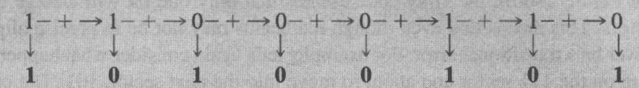
EXAMPLE 2-37

(a) Convert the binary number 11000110 to Gray code.

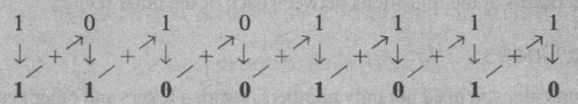
(b) Convert the Gray code 10101111 to binary.

Solution

(a) Binary to Gray code:



(b) Gray code to binary:



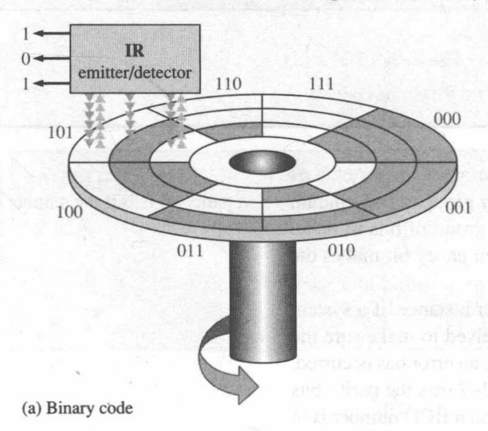
Related Problem

(a) Convert binary 101101 to Gray code.

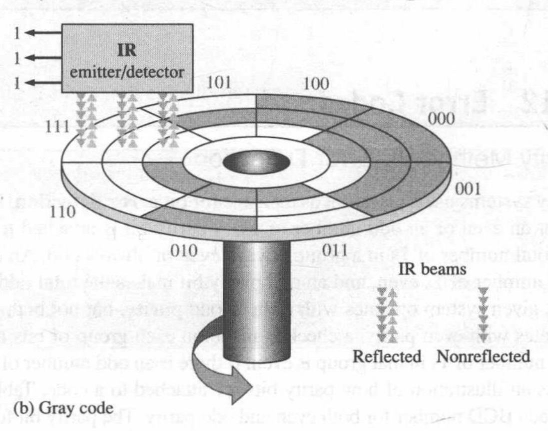
(b) Convert Gray code 100111 to binary.

An Application

The concept of a 3-bit shaft position encoder is shown in Figure 2–6. Basically, there are three concentric rings that are segmented into eight sectors. The more sectors there are, the more accurately the position can be represented, but we are using only eight to illustrate. Each sector of each ring is either reflective or nonreflective. As the rings rotate with the shaft, they come under an IR emitter that produces three separate IR beams. A 1 is indicated where there is a reflected beam, and a 0 is indicated where there is no reflected beam. The IR detector senses the presence or absence of reflected beams and produces a corresponding 3-bit code. The IR emitter/detector is in a fixed position. As the shaft rotates counterclockwise through 360°, the eight sectors move under the three beams. Each beam is either reflected or absorbed by the sector surface to represent a binary or Gray code number that indicates the shaft position.



(a) Binary code



(b) Gray code

▲FIGURE 2-6

A simplified illustration of how the Gray code solves the error problem in shaft position encoders. Three bits are shown to illustrate the concept, although most shaft encoders use more than 10 bits to achieve a higher resolution.

In Figure 2–6(a), the sectors are arranged in a straight binary pattern, so that the detector output goes from 000 to 001 to 010 to 011 and so on. When a beam is aligned over a reflective sector, the output is 1; when a beam is aligned over a nonreflective sector, the output is 0. If one beam is slightly ahead of the others during the transition from one sector to the next, an

erroneous output can occur. Consider what happens when the beams are on the 111 sector and about to enter the 000 sector. If the MSB beam is slightly ahead, the position would be incorrectly indicated by a transitional 011 instead of a 111 or a 000. In this type of application, it is virtually impossible to maintain precise mechanical alignment of the IR emitter/detector beams; therefore, some error will usually occur at many of the transitions between sectors.

The Gray code is used to eliminate the error problem which is inherent in the binary code. As shown in Figure 2–6(b), the Gray code assures that only one bit will change between adjacent sectors. This means that even though the beams may not be in precise alignment, there will never be a transitional error. For example, let's again consider what happens when the beams are on the 111 sector and about to move into the next sector, 101. The only two possible outputs during the transition are 111 and 101, no matter how the beams are aligned. A similar situation occurs at the transitions between each of the other sectors.

Alphanumeric Codes

In order to communicate, you need not only numbers, but also letters and other symbols. In the strictest sense, **alphanumeric** codes are codes that represent numbers and alphabetic characters (letters). Most such codes, however, also represent other characters such as symbols and various instructions necessary for conveying information.

At a minimum, an alphanumeric code must represent 10 decimal digits and 26 letters of the alphabet, for a total of 36 items. This number requires six bits in each code combination because five bits are insufficient ($2^5 = 32$). There are 64 total combinations of six bits, so there are 28 unused code combinations. Obviously, in many applications, symbols other than just numbers and letters are necessary to communicate completely. You need spaces, periods, colons, semicolons, question marks, etc. You also need instructions to tell the receiving system what to do with the information. With codes that are six bits long, you can handle decimal numbers, the alphabet, and 28 other symbols. This should give you an idea of the requirements for a basic alphanumeric code. The ASCII is a common alphanumeric code and is covered next.

SECTION 2-11 CHECKUP

1. Convert the following binary numbers to the Gray code:
   (a) 1100 (b) 1010 (c) 11010
2. Convert the following Gray codes to binary:
   (a) 1000 (b) 1010 (c) 11101

2-12 Error Codes

Parity Method for Error Detection

Many systems use a parity bit as a means for bit **error detection**. Any group of bits contain either an even or an odd number of 1s. A parity bit is attached to a group of bits to make the total number of 1s in a group always even or always odd. An even parity bit makes the total number of 1s even, and an odd parity bit makes the total odd.

A parity bit tells if the number of 1s is odd or even.

A given system operates with even or odd **parity**, but not both. For instance, if a system operates with even parity, a check is made on each group of bits received to make sure the total number of 1s in that group is even. If there is an odd number of 1s, an error has occurred.

As an illustration of how parity bits are attached to a code, Table 2–7 lists the parity bits for each BCD number for both even and odd parity. The parity bit for each BCD number is in the *P* column.

The parity bit can be attached to the code at either the beginning or the end, depending on system design. Notice that the total number of 1s, including the parity bit, is always even for even parity and always odd for odd parity.

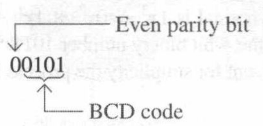
TABLE 2–7

The BCD code with parity bits.

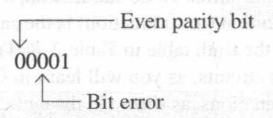
Even Parity		Odd Parity	
P	BCD	*P*	BCD
0	0000	1	0000
1	0001	0	0001
1	0010	0	0010
0	0011	1	0011
1	0100	0	0100
0	0101	1	0101
0	0110	1	0110
1	0111	0	0111
1	1000	0	1000
0	1001	1	1001

Detecting an Error

A parity bit provides for the detection of a single bit error (or any odd number of errors, which is very unlikely) but cannot check for two errors in one group. For instance, let's assume that we wish to transmit the BCD code 0101. (Parity can be used with any number of bits; we are using four for illustration.) The total code transmitted, including the even parity bit, is



Now let's assume that an error occurs in the third bit from the left (the 1 becomes a 0).



When this code is received, the parity check circuitry determines that there is only a single 1 (odd number), when there should be an even number of 1s. Because an even number of 1s does not appear in the code when it is received, an error is indicated.

An odd parity bit also provides in a similar manner for the detection of a single error in a given group of bits.

EXAMPLE 2-38

Assign the proper even parity bit to the following code groups:

(a) 1010 (b) 111000 (c) 101101
(d) 1000111001001 (e) 101101011111

Solution

Make the parity bit either 1 or 0 as necessary to make the total number of 1s even. The parity bit will be the left-most bit (color).

(a) 01010 (b) 1111000 (c) 0101101
(d) 01000111001001 (e) 1101101011111

Related Problem

Add an even parity bit to the 7-bit ASCII code for the letter K.

EXAMPLE 2-39

An odd parity system receives the following code groups: 10110, 11010, 110011, 110101110100, and 1100010101010. Determine which groups, if any, are in error.

Solution

Since odd parity is required, any group with an even number of 1s is incorrect. The following groups are in error: **110011** and **1100010101010**.

Cyclic Redundancy Check

The **cyclic redundancy check (CRC)** is a widely used code used for detecting one- and two-bit transmission errors when digital data are transferred on a communication link. The communication link can be between two computers that are connected to a network or between a digital storage device (such as a CD, DVD, or a hard drive) and a PC. If it is properly designed, the CRC can also detect multiple errors for a number of bits in sequence (burst errors). In CRC, a certain number of check bits, sometimes called a *checksum*, are

appended to the data bits (added to end) that are being transmitted. The transmitted data are tested by the receiver for errors using the CRC. Not every possible error can be identified, but the CRC is much more efficient than just a simple parity check.

CRC is often described mathematically as the division of two polynomials to generate a remainder. A polynomial is a mathematical expression that is a sum of terms with positive exponents. When the coefficients are limited to 1s and 0s, it is called a *univariate polynomial.* An example of a univariate polynomial is $1x^3 + 0x^2 + 1x^1 + 1x^0$ or simply $x^3 + x^1 + x^0$, which can be fully described by the 4-bit binary number 1011. Most cyclic redundancy checks use a 16-bit or larger polynomial, but for simplicity the process is illustrated here with four bits.

Modulo-2 Operations

Simply put, CRC is based on the division of two binary numbers; and, as you know, division is just a series of subtractions and shifts. To do subtraction, a method called *modulo-2* addition can be used. Modulo-2 addition (or subtraction) is the same as binary addition with the carries discarded, as shown in the truth table in Table 2–8. **Truth tables** are widely used to describe the operation of logic circuits, as you will learn in Chapter 3. With two bits, there is a total of four possible combinations, as shown in the table. This particular table describes the modulo-2 operation also known as *exclusive-OR* and can be implemented with a logic gate that will be introduced in Chapter 3. A simple rule for modulo-2 is that the output is 1 if the inputs are different; otherwise, it is 0.

TABLE 2–8

Modulo-2 operation.

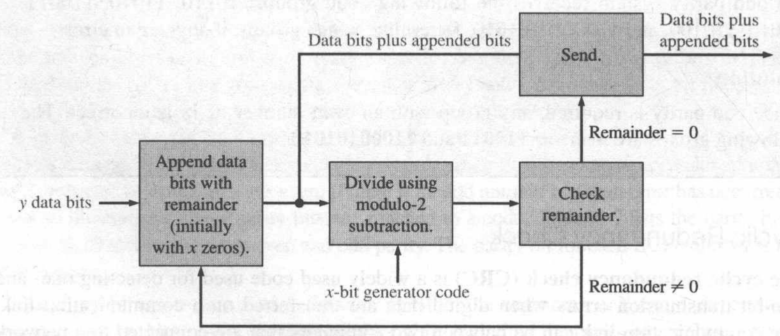
Input Bits	Output Bit
0 0	0
0 1	1
1 0	1
1 1	0

CRC Process

The process is as follows:

1. Select a fixed generator code; it can have fewer bits than the data bits to be checked. This code is understood in advance by both the sending and receiving devices and must be the same for both.
2. Append a number of 0s equal to the number of bits in the generator code to the data bits.
3. Divide the data bits including the appended bits by the generator code bits using modulo-2.
4. If the remainder is 0, the data and appended bits are sent as is.
5. If the remainder is not 0, the appended bits are made equal to the remainder bits in order to get a 0 remainder before data are sent.
6. At the receiving end, the receiver divides the incoming appended data bit code by the same generator code as used by the sender.
7. If the remainder is 0, there is no error detected (it is possible in rare cases for multiple errors to cancel). If the remainder is not 0, an error has been detected in the transmission and a retransmission is requested by the receiver.

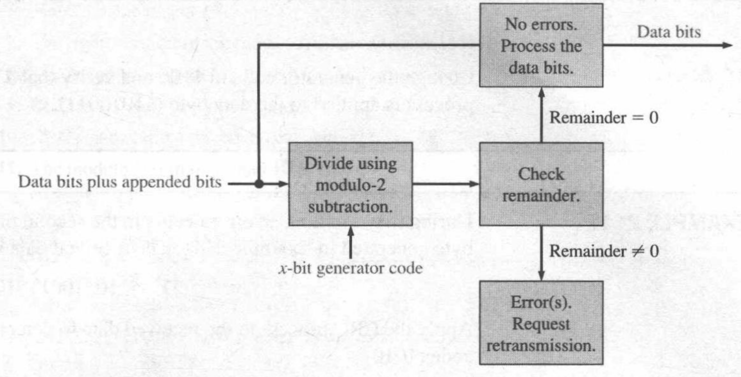
Figure 2–7 illustrates the CRC process.



(a) Transmitting end of communication link

▶ FIGURE 2-7

The CRC process.



(b) Receiving end of communication link

EXAMPLE 2-40

Determine the transmitted CRC for the following byte of data (D) and generator code (G). Verify that the remainder is 0.

D: 11010011

G: 1010

Solution

Since the generator code has four data bits, add four 0s (blue) to the data byte. The appended data (D′) is

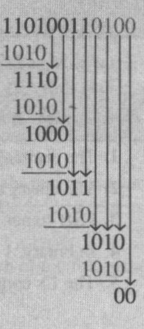
$$D' = 110100110000$$

Divide the appended data by the generator code (red) using the modulo-2 operation until all bits have been used.

$$\frac{D'}{G} = \frac{110100110000}{1010}$$

```
110100110000
1010
 1110
 1010
  1000
  1010
    1011
    1010
      1000
      1010
        100
```

Remainder = 0100. Since the remainder is not 0, append the data with the four remainder bits (blue). Then divide by the generator code (red). The transmitted CRC is **110100110100**.



Remainder = 0

Related Problem

Change the generator code to 1100 and verify that a 0 remainder results when the CRC process is applied to the data byte (11010011).

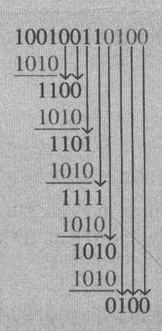
EXAMPLE 2-41

During transmission, an error occurs in the second bit from the left in the appended data byte generated in Example 2–41. The received data is

$$D' = 100100110100$$

Apply the CRC process to the received data to detect the error using the same generator code (1010).

Solution



Remainder = 0100. Since it is not zero, an error is indicated.

Related Problem

Assume two errors in the data byte as follows: 10011011. Apply the CRC process to check for the errors using the same received data and the same generator code.

SECTION 2-12 CHECKUP

1. Which odd-parity code is in error?
   (a) 1011 **(b)** 1110 **(c)** 0101 **(d)** 1000
2. Which even-parity code is in error?
   (a) 11000110 **(b)** 00101000 **(c)** 10101010 **(d)** 11111011
3. Add an even parity bit to the end of each of the following codes.
   (a) 1010100 **(b)** 0100000 **(c)** 1110111 **(d)** 1000110
4. What does CRC stand for?
5. Apply modulo-2 operations to determine the following:
   (a) $1 + 1$ **(b)** $1 - 1$ **(c)** $1 - 0$ **(d)** $0 + 1$

TRUE/FALSE QUIZ

Answers are at the end of the chapter.

1. The decimal number system is a weighted system with ten digits.
2. The binary number system is a weighted system with two digits.
3. LSB stands for lowest single bit.
4. In binary, $1 + 1 = 2$.
5. The 1's complement of the binary number 1010 is 0101.

6. The 2's complement of the binary number 0001 is 1110.

7. The right-most bit in a signed binary number is the sign bit.

8. The hexadecimal number system has 16 characters, six of which are alphabetic characters.

9. BCD stands for binary coded decimal.

10. CRC stands for cyclic redundancy check.

11. The modulo-2 sum of 11 and 10 is 100.

SELF-TEST

Answers are at the end of the chapter.

1. $2 \times 10^1 + 8 \times 10^0$ is equal to
(a) 10 **(b)** 280 **(c)** 2.8 **(d)** 28

2. The binary number 1101 is equal to the decimal number
(a) 13 **(b)** 49 **(c)** 11 **(d)** 3

3. The binary number 11011101 is equal to the decimal number
(a) 121 **(b)** 221 **(c)** 441 **(d)** 256

4. The decimal number 17 is equal to the binary number
(a) 10010 **(b)** 11000 **(c)** 10001 **(d)** 01001

5. The decimal number 175 is equal to the binary number
(a) 11001111 **(b)** 10101110 **(c)** 10101111 **(d)** 11101111

6. The sum of 11010 + 01111 equals
(a) 101001 **(b)** 101010 **(c)** 110101 **(d)** 101000

7. The difference of 110 − 010 equals
(a) 001 **(b)** 010 **(c)** 101 **(d)** 100

8. The 1's complement of 10111001 is
(a) 01000111 **(b)** 01000110 **(c)** 11000110 **(d)** 10101010

9. The 2's complement of 11001000 is
(a) 00110111 **(b)** 00110001 **(c)** 01001000 **(d)** 00111000

10. The decimal number +122 is expressed in the 2's complement form as
(a) 01111010 **(b)** 11111010 **(c)** 01000101 **(d)** 10000101

11. The decimal number −34 is expressed in the 2's complement form as
(a) 01011110 **(b)** 10100010 **(c)** 11011110 **(d)** 01011101

12. A single-precision floating-point binary number has a total of
(a) 8 bits **(b)** 16 bits **(c)** 24 bits **(d)** 32 bits

13. In the 2's complement form, the binary number 10010011 is equal to the decimal number
(a) −19 **(b)** +109 **(c)** +91 **(d)** −109

14. The binary number 101100111001010100001 can be written in octal as
(a) 5471230_8 **(b)** 5471241_8 **(c)** 2634521_8 **(d)** 23162501_8

15. The binary number 10001101010001101111 can be written in hexadecimal as
(a) $AD467_{16}$ **(b)** $8C46F_{16}$ **(c)** $8D46F_{16}$ **(d)** $AE46F_{16}$

16. The binary number for $F7A9_{16}$ is
(a) 1111011110101001 **(b)** 1110111110101001
(c) 1111111010110001 **(d)** 1111011010101001

17. The BCD number for decimal 473 is
(a) 111011010 **(b)** 110001110011 **(c)** 010001110011 **(d)** 010011110011

18. The code that has an even-parity error is
(a) 1010011 **(b)** 1101000 **(c)** 1001000 **(d)** 1110111

19. In the cyclic redundancy check, the absence of errors is indicated by
(a) Remainder = generator code **(b)** Remainder = 0
(c) Remainder = 1 **(d)** Quotient = 0

PROBLEMS

Answers to odd-numbered problems are at the end of the book.

Section 2–1 Decimal Numbers

1. What is the weight of the digit 6 in each of the following decimal numbers?
(a) 1386 **(b)** 54,692 **(c)** 671,920

2. Express each of the following decimal numbers as a power of ten:

(a) 10 **(b)** 100 **(c)** 10,000 **d)** 1,000,000

3. Give the value of each digit in the following decimal numbers:

(a) 471 **(b)** 9356 **(c)** 125,000

4. How high can you count with four decimal digits?

Section 2–2 Binary Numbers

5. Convert the following binary numbers to decimal:

(a) 11 **(b)** 100 **(c)** 111 **(d)** 1000
(e) 1001 **(f)** 1100 **(g)** 1011 **(h)** 1111

6. Convert the following binary numbers to decimal:

(a) 1110 **(b)** 1010 **(c)** 11100 **(d)** 10000
(e) 10101 **(f)** 11101 **(g)** 10111 **(h)** 11111

7. Convert each binary number to decimal:

(a) 110011.11 **(b)** 101010.01 **(c)** 1000001.111
(d) 1111000.101 **(e)** 1011100.10101 **(f)** 1110001.0001
(g) 1011010.1010 **(h)** 1111111.11111

8. What is the highest decimal number that can be represented by each of the following numbers of binary digits (bits)?

(a) two **(b)** three **(c)** four **(d)** five **(e)** six
(f) seven **(g)** eight **(h)** nine **(i)** ten **(j)** eleven

9. How many bits are required to represent the following decimal numbers?

(a) 17 **(b)** 35 **(c)** 49 **(d)** 68
(e) 81 **(f)** 114 **(g)** 132 **(h)** 205

10. Generate the binary sequence for each decimal sequence:

(a) 0 through 7 **(b)** 8 through 15 **(c)** 16 through 31
(d) 32 through 63 **(e)** 64 through 75

Section 2–3 Decimal-to-Binary Conversion

11. Convert each decimal number to binary by using the sum-of-weights method:

(a) 10 **(b)** 17 **(c)** 24 **(d)** 48
(e) 61 **(f)** 93 **(g)** 125 **(h)** 186

12. Convert each decimal fraction to binary using the sum-of-weights method:

(a) 0.32 **(b)** 0.246 **(c)** 0.0981

13. Convert each decimal number to binary using repeated division by 2:

(a) 15 **(b)** 21 **(c)** 28 **(d)** 34
(e) 40 **(f)** 59 **(g)** 65 **(h)** 73

14. Convert each decimal fraction to binary using repeated multiplication by 2:

(a) 0.98 **(b)** 0.347 **(c)** 0.9028

Section 2–4 Binary Arithmetic

15. Add the binary numbers:

(a) 11 + 01 **(b)** 10 + 10 **(c)** 101 + 11
(d) 111 + 110 **(e)** 1001 + 101 **(f)** 1101 + 1011

16. Use direct subtraction on the following binary numbers:

(a) 11 − 1 **(b)** 101 − 100 **(c)** 110 − 101
(d) 1110 − 11 **(e)** 1100 − 1001 **(f)** 11010 − 10111

17. Perform the following binary multiplications:

(a) 11 × 11 **(b)** 100 × 10 **(c)** 111 × 101
(d) 1001 × 110 **(e)** 1101 × 1101 **(f)** 1110 × 1101

18. Divide the binary numbers as indicated:

(a) 100 ÷ 10 **(b)** 1001 ÷ 11 **(c)** 1100 ÷ 100

Section 2–5 Complements of Binary Numbers

19. What are two ways of representing zero in 1's complement form?

20. How is zero represented in 2's complement form?

21. Determine the 1's complement of each binary number:

(a) 101 **(b)** 110 **(c)** 1010
(d) 11010111 **(e)** 1110101 **(f)** 00001

22. Determine the 2's complement of each binary number using either method:

(a) 10 **(b)** 111 **(c)** 1001 **(d)** 1101
(e) 11100 **(f)** 10011 **(g)** 10110000 **(h)** 00111101

Section 2–6 Signed Numbers

23. Express each decimal number in binary as an 8-bit sign-magnitude number:

(a) +29 **(b)** −85 **(c)** +100 **(d)** −123

24. Express each decimal number as an 8-bit number in the 1's complement form:

(a) −34 **(b)** +57 **(c)** −99 **(d)** +115

25. Express each decimal number as an 8-bit number in the 2's complement form:

(a) +12 **(b)** −68 **(c)** +101 **(d)** −125

26. Determine the decimal value of each signed binary number in the sign-magnitude form:

(a) 10011001 **(b)** 01110100 **(c)** 10111111

27. Determine the decimal value of each signed binary number in the 1's complement form:

(a) 10011001 **(b)** 01110100 **(c)** 10111111

28. Determine the decimal value of each signed binary number in the 2's complement form:

(a) 10011001 **(b)** 01110100 **(c)** 10111111

29. Express each of the following sign-magnitude binary numbers in single-precision floating-point format:

(a) 0111110000101011 **(b)** 100110000011000

30. Determine the values of the following single-precision floating-point numbers:

(a) 1 10000001 01001001110001000000000
(b) 0 11001100 10000111110100100000000

Section 2–7 Arithmetic Operations with Signed Numbers

31. Convert each pair of decimal numbers to binary and add using the 2's complement form:

(a) 33 and 15 **(b)** 56 and −27 **(c)** −46 and 25 **(d)** −110 and −84

32. Perform each addition in the 2's complement form:

(a) 00010110 + 00110011 **(b)** 01110000 + 10101111

33. Perform each addition in the 2's complement form:

(a) 10001100 + 00111001 **(b)** 11011001 + 11100111

34. Perform each subtraction in the 2's complement form:

(a) 00110011 − 00010000 **(b)** 01100101 − 11101000

35. Multiply 01101010 by 11110001 in the 2's complement form.

36. Divide 01000100 by 00011001 in the 2's complement form.

Section 2–8 Hexadecimal Numbers

37. Convert each hexadecimal number to binary:

(a) 38_{16} **(b)** 59_{16} **(c)** $A14_{16}$ **(d)** $5C8_{16}$
(e) 4100_{16} **(f)** $FB17_{16}$ **(g)** $8A9D_{16}$

38. Convert each binary number to hexadecimal:

(a) 1110 **(b)** 10 **(c)** 10111
(d) 10100110 **(e)** 1111110000 **(f)** 100110000010

39. Convert each hexadecimal number to decimal:

(a) 23_{16} **(b)** 92_{16} **(c)** $1A_{16}$ **(d)** $8D_{16}$
(e) $F3_{16}$ **(f)** EB_{16} **(g)** $5C2_{16}$ **(h)** 700_{16}

40. Convert each decimal number to hexadecimal:

(a) 8 **(b)** 14 **(c)** 33 **(d)** 52
(e) 284 **(f)** 2890 **(g)** 4019 **(h)** 6500

41. Perform the following additions:

(a) $37_{16} + 29_{16}$ **(b)** $A0_{16} + 6B_{16}$ **(c)** $FF_{16} + BB_{16}$

42. Perform the following subtractions:

(a) $51_{16} - 40_{16}$ **(b)** $C8_{16} - 3A_{16}$ **(c)** $FD_{16} - 88_{16}$

Section 2–9 Octal Numbers

43. Convert each octal number to decimal:

(a) 12_8 **(b)** 27_8 **(c)** 56_8 **(d)** 64_8 **(e)** 103_8
(f) 557_8 **(g)** 163_8 **(h)** 1024_8 **(i)** 7765_8

44. Convert each decimal number to octal by repeated division by 8:

(a) 15 **(b)** 27 **(c)** 46 **(d)** 70
(e) 100 **(f)** 142 **(g)** 219 **(h)** 435

45. Convert each octal number to binary:

(a) 13_8 **(b)** 57_8 **(c)** 101_8 **(d)** 321_8 **(e)** 540_8
(f) 4653_8 **(g)** 13271_8 **(h)** 45600_8 **(i)** 100213_8

46. Convert each binary number to octal:

(a) 111 **(b)** 10 **(c)** 110111
(d) 101010 **(e)** 1100 **(f)** 1011110
(g) 101100011001 **(h)** 10110000011 **(i)** 111111101111000

Section 2–10 Binary Coded Decimal (BCD)

47. Convert each of the following decimal numbers to 8421 BCD:

(a) 10 **(b)** 13 **(c)** 18 **(d)** 21 **(e)** 25 **(f)** 36
(g) 44 **(h)** 57 **(i)** 69 **(j)** 98 **(k)** 125 **(l)** 156

48. Convert each of the decimal numbers in Problem 47 to straight binary, and compare the number of bits required with that required for BCD.

49. Convert the following decimal numbers to BCD:

(a) 104 **(b)** 128 **(c)** 132 **(d)** 150 **(e)** 186
(f) 210 **(g)** 359 **(h)** 547 **(i)** 1051

50. Convert each of the BCD numbers to decimal:

(a) 0001 **(b)** 0110 **(c)** 1001
(d) 00011000 **(e)** 00011001 **(f)** 00110010
(g) 01000101 **(h)** 10011000 **(i)** 100001110000

51. Convert each of the BCD numbers to decimal:

(a) 10000000 **(b)** 001000110111
(c) 001101000110 **(d)** 010000100001
(e) 011101010100 **(f)** 100000000000
(g) 100101111000 **(h)** 0001011010000011
(i) 1001000000011000 **(j)** 0110011001100111

52. Add the following BCD numbers:

(a) 0010 + 0001 **(b)** 0101 + 0011
(c) 0111 + 0010 **(d)** 1000 + 0001
(e) 00011000 + 00010001 **(f)** 01100100 + 00110011
(g) 01000000 + 01000111 **(h)** 10000101 + 00010011

53. Add the following BCD numbers:

(a) 1000 + 0110 **(b)** 0111 + 0101
(c) 1001 + 1000 **(d)** 1001 + 0111
(e) 00100101 + 00100111 **(f)** 01010001 + 01011000
(g) 10011000 + 10010111 **(h)** 010101100001 + 011100001000

54. Convert each pair of decimal numbers to BCD, and add as indicated:

(a) 4 + 3 **(b)** 5 + 2 **(c)** 6 + 4 **(d)** 17 + 12
(e) 28 + 23 **(f)** 65 + 58 **(g)** 113 + 101 **(h)** 295 + 157

Section 2–11 Digital Codes

55. In a certain application a 4-bit binary sequence cycles from 1111 to 0000 periodically. There are four bit changes, and because of circuit delays, these changes may not occur at the same instant. For example, if the LSB changes first, the number will appear as 1110 during the transition from 1111 to 0000 and may be misinterpreted by the system. Illustrate how the Gray code avoids this problem.

56. Convert each binary number to Gray code:

(a) 11011 **(b)** 1001010 **(c)** 1111011101110

57. Convert each Gray code to binary:

(a) 1010 **(b)** 00010 **(c)** 11000010001

Section 2–12 Error Codes

58. Determine which of the following even parity codes are in error:

(a) 100110010 **(b)** 011101010 **(c)** 10111111010001010

59. Determine which of the following odd parity codes are in error:

(a) 11110110 **(b)** 00110001 **(c)** 01010101010101010

60. Attach the proper even parity bit to each of the following bytes of data:

(a) 10100100 **(b)** 00001001 **(c)** 11111110

61. Apply modulo-2 to the following:

(a) 1100 + 1011 **(b)** 1111 + 0100 **(c)** 10011001 + 100011100

62. Verify that modulo-2 subtraction is the same as modulo-2 addition by adding the result of each operation in problem 66 to either of the original numbers to get the other number. This will show that the result is the same as the difference of the two numbers.

63. Apply CRC to the data bits 10110010 using the generator code 1010 to produce the transmitted CRC code.

64. Assume that the code produced in problem 68 incurs an error in the most significant bit during transmission. Apply CRC to detect the error.

ANSWERS

SECTION CHECKUPS

Section 2–1 Decimal Numbers

1. (a) 1370: 10 **(b)** 6725: 100 **(c)** 7051: 1000 **(d)** 58.72: 0.1

2. (a) $51 = (5 \times 10) + (1 \times 1)$

(b) $137 = (1 \times 100) + (3 \times 10) + (7 \times 1)$

(c) $1492 = (1 \times 1000) + (4 \times 100) + (9 \times 10) + (2 \times 1)$

(d) $106.58 = (1 \times 100) + (0 \times 10) + (6 \times 1) + (5 \times 0.1) + (8 \times 0.01)$

Section 2–2 Binary Numbers

1. $2^8 - 1 = 255$

2. Weight is 16.

3. 10111101.011 = 189.375

Section 2–3 Decimal-to-Binary Conversion

1. (a) 23 = 10111 **(b)** 57 = 111001 **(c)** 45.5 = 101101.1

2. (a) 14 = 1110 **(b)** 21 = 10101 **(c)** 0.375 = 0.011

Section 2–4 Binary Arithmetic

1. (a) 1101 + 1010 = 10111 **(b)** 10111 + 01101 = 100100

2. (a) 1101 − 0100 = 1001 **(b)** 1001 − 0111 = 0010

3. (a) $110 \times 111 = 101010$ **(b)** $1100 \div 011 = 100$

Section 2–5 Complements of Binary Numbers

1. **(a)** 1's comp of 00011010 = 11100101 **(b)** 1's comp of 11110111 = 00001000
   (c) 1's comp of 10001101 = 01110010
2. **(a)** 2's comp of 00010110 = 11101010 **(b)** 2's comp of 11111100 = 00000100
   (c) 2's comp of 10010001 = 01101111

Section 2–6 Signed Numbers

1. Sign-magnitude: +9 = 00001001
2. 1's comp: −33 = 11011110
3. 2's comp: −46 = 11010010
4. Sign bit, exponent, and mantissa

Section 2–7 Arithmetic Operations with Signed Numbers

1. Cases of addition: positive number is larger, negative number is larger, both are positive, bo are negative
2. 00100001 + 10111100 = 11011101
3. 01110111 − 00110010 = 01000101
4. Sign of product is positive.
5. 00000101 × 01111111 = 01001111011
6. Sign of quotient is negative.
7. 00110000 ÷ 00001100 = 00000100

Section 2–8 Hexadecimal Numbers

1. **(a)** 10110011 = $B3_{16}$ **(b)** 110011101000 = $CE8_{16}$
2. **(a)** 57_{16} = 01010111 **(b)** $3A5_{16}$ = 001110100101
   (c) $F8OB_{16}$ = 1111100000001011
3. $9B30_{16} = 39{,}728_{10}$
4. $573_{10} = 23D_{16}$
5. **(a)** $18_{16} + 34_{16} = 4C_{16}$ **(b)** $3F_{16} + 2A_{16} = 69_{16}$
6. **(a)** $75_{16} - 21_{16} = 54_{16}$ **(b)** $94_{16} - 5C_{16} = 38_{16}$

Section 2–9 Octal Numbers

1. **(a)** $73_8 = 59_{10}$ **(b)** $125_8 = 85_{10}$
2. **(a)** $98_{10} = 142_8$ **(b)** $163_{10} = 243_8$
3. **(a)** 46_8 = 100110 **(b)** 723_8 = 111010011 **(c)** 5624_8 = 101110010100
4. **(a)** 110101111 = 657_8 **(b)** 1001100010 = 1142_8 **(c)** 10111111001 = 2771_8

Section 2–10 Binary Coded Decimal (BCD)

1. **(a)** 0010: 2 **(b)** 1000: 8 **(c)** 0001: 1 **(d)** 0100: 4
2. **(a)** 6_{10} = 0110 **(b)** 15_{10} = 00010101 **(c)** 273_{10} = 001001110011
   (d) 849_{10} = 100001001001
3. **(a)** 10001001 = 89_{10} **(b)** 001001111000 = 278_{10} **(c)** 000101010111 = 157_{10}
4. A 4-bit sum is invalid when it is greater than 9_{10}.

Section 2–11 Digital Codes

1. **(a)** 1100_2 = 1010 Gray **(b)** 1010_2 = 1111 Gray **(c)** 11010_2 = 10111 Gray
2. **(a)** 1000 Gray = 1111_2 **(b)** 1010 Gray = 1100_2 **(c)** 11101 Gray = 10110_2

Section 2–12 Error Codes

1. **(c)** 0101 has an error.
2. **(d)** 11111011 has an error.

3. (a) 10101001 **(b)** 01000001 **(c)** 11101110 **(d)** 10001101

4. Cyclic redundancy check

5. (a) 0 **(b)** 0 **(c)** 1 **(d)** 1

RELATED PROBLEMS FOR EXAMPLES

2–1 9 has a value of 900, 3 has a value of 30, 9 has a value of 9.

2–2 6 has a value of 60, 7 has a value of 7, 9 has a value of 9/10 (0.9), 2 has a value of 2/100 (0.02), 4 has a value of 4/1000 (0.004).

2–3 $10010001 = 128 + 16 + 1 = 145$

2–4 $10.111 = 2 + 0.5 + 0.25 + 0.125 = 2.875$

2–5 $125 = 64 + 32 + 16 + 8 + 4 + 1 = 1111101$

2–6 $39 = 100111$

2–7 $1111 + 1100 = 11011$

2–8 $111 - 100 = 011$

2–9 $110 - 101 = 001$

2–10 $1101 \times 1010 = 10000010$

2–11 $1100 \div 100 = 11$

2–12 00110101

2–13 01000000

2–14 See Table 2–9.

▶ **TABLE 2–9**

	Sign-Magnitude	1's Comp	2's Comp
+19	00010011	00010011	00010011
−19	10010011	11101100	11101101

2–15 $01110111 = +119_{10}$

2–16 $11101011 = -20_{10}$

2–17 $11010111 = -41_{10}$

2–18 11000010001010011000000000

2–19 01010101

2–20 00010001

2–21 1001000110

2–22 $(83)(-59) = -4897$ (10110011011111 in 2's comp)

2–23 $100 \div 25 = 4$ (0100)

2–24 $4F79C_{16}$

2–25 0110101111010011_2

2–26 $6BD_{16} = 011010111101 = 2^{10} + 2^9 + 2^7 + 2^5 + 2^4 + 2^3 + 2^2 + 2^0$
$= 1024 + 512 + 128 + 32 + 16 + 8 + 4 + 1 = 1725_{10}$

2–27 $60A_{16} = (6 \times 256) + (0 \times 16) + (10 \times 1) = 1546_{10}$

2–28 $2591_{10} = A1F_{16}$

2–29 $4C_{16} + 3A_{16} = 86_{16}$

2–30 $BCD_{16} - 173_{16} = A5A_{16}$

2–31 (a) $001011_2 = 11_{10} = 13_8$ **(b)** $010101_2 = 21_{10} = 25_8$
(c) $001100000_2 = 96_{10} = 140_8$ **(d)** $111101010110_2 = 3926_{10} = 7526_8$

2–32 1250762_8

2–33 1001011001110011

2–34 $82{,}276_{10}$

2–35 1001100101101000

2–36 10000010

2–37 **(a)** 111011 (Gray) **(b)** 111010_2

2–38 01001011

2–39 Yes

2–40 A 0 remainder results

2–41 Errors are indicated.

TRUE/FALSE QUIZ

1. T **2.** T **3.** F **4.** F **5.** T **6.** F **7.** F **8.** T **9.** T **10.** T
11. F

SELF-TEST

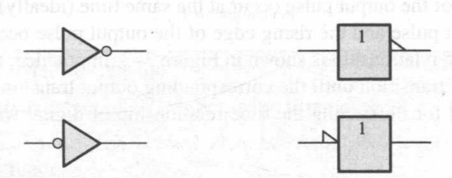
1. (d) **2.** (a) **3.** (b) **4.** (c) **5.** (c) **6.** (a) **7.** (d) **8.** (b)
9. (d) **10.** (a) **11.** (c) **12.** (d) **13.** (d) **14.** (b) **15.** (c) **16.** (a)
17. (c) **18.** (b) **19.** (b)

Chapter 3 Logic Gates

CHAPTER OUTLINE

3-1 The Inverter

Standard logic symbols for the **inverter** are shown in Figure 3–1. Part (a) shows the *distinctive shape* symbols, and part (b) shows the *rectangular outline* symbols. In this textbook, distinctive shape symbols are generally used; however, the rectangular outline symbols are found in many industry publications, and you should become familiar with them as well. (Logic symbols are in accordance with **ANSI/IEEE** Standard 91-1984 and its supplement Standard 91a-1991.)



(a) Distinctive shape symbols with negation indicators

(b) Rectangular outline symbols with polarity indicators

▶ **FIGURE 3-1**

Standard logic symbols for the inverter (ANSI/IEEE Std. 91-1984/Std. 91a-1991).

The Negation and Polarity Indicators

The negation indicator is a "bubble" (○) that indicates **inversion** or *complementation* when it appears on the input or output of any logic element, as shown in Figure 3–1(a) for the inverter. Generally, inputs are on the left of a logic symbol and the output is on the right. When appearing on the input, the bubble means that a 0 is the active or *asserted* input state, and the input is called an active-LOW input. When appearing on the output, the bubble means that a 0 is the active or asserted output state, and the output is called an active-LOW output. The absence of a bubble on the input or output means that a 1 is the active or asserted state, and in this case, the input or output is called active-HIGH.

The polarity or level indicator is a "triangle" (◺) that indicates inversion when it appears on the input or output of a logic element, as shown in Figure 3–1(b). When appearing on the input, it means that a LOW level is the active or asserted input state. When appearing on the output, it means that a LOW level is the active or asserted output state.

Either indicator (bubble or triangle) can be used both on distinctive shape symbols and on rectangular outline symbols. Figure 3–1(a) indicates the principal inverter symbols used in this text. Note that a change in the placement of the negation or polarity indicator does not imply a change in the way an inverter operates.

Inverter Truth Table

When a HIGH level is applied to an inverter input, a LOW level will appear on its output. When a LOW level is applied to its input, a HIGH will appear on its output. This operation is summarized in Table 3–1, which shows the output for each possible input in terms of levels and corresponding bits. A table such as this is called a **truth table**.

▼**TABLE 3-1**

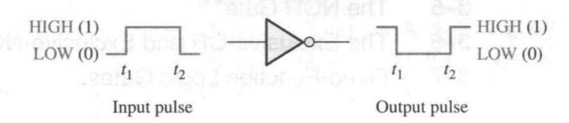
Inverter truth table.

Input	Output
LOW (0)	HIGH (1)
HIGH (1)	LOW (0)

Inverter Operation

Figure 3–2 shows the output of an inverter for a pulse input, where t_1 and t_2 indicate the corresponding points on the input and output pulse waveforms.

When the input is LOW, the output is HIGH; when the input is HIGH, the output is LOW, thereby producing an inverted output pulse.



▲**FIGURE 3-2**

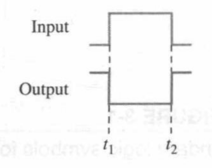
Inverter operation with a pulse input.Open file F03-02 to verify inverter operation. *A Multisim tutorial is available on the website.*

MultiSim

Timing Diagrams

Recall from Chapter 1 that a *timing diagram* is basically a graph that accurately displays the relationship of two or more waveforms with respect to each other on a time basis. For example, the time relationship of the output pulse to the input pulse in Figure 3–2 can be shown with a simple timing diagram by aligning the two pulses so that the occurrences of the pulse edges appear in the proper time relationship. The rising edge of the input pulse and the falling edge of the output pulse occur at the same time (ideally). Similarly, the falling edge of the input pulse and the rising edge of the output pulse occur at the same time (ideally). This timing relationship is shown in Figure 3–3. In practice, there is a very small delay from the input transition until the corresponding output transition. Timing diagrams are especially useful for illustrating the time relationship of digital waveforms with multiple pulses.

A timing diagram shows how two or more waveforms relate in time.



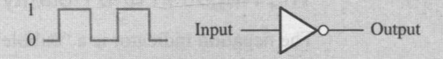
▲**FIGURE 3-3**

Timing diagram for the case in Figure 3–2.

EXAMPLE 3-1

A waveform is applied to an inverter in Figure 3–4. Determine the output waveform corresponding to the input and show the timing diagram. According to the placement of the bubble, what is the active output state?

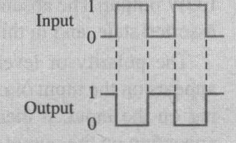
►**FIGURE 3-4**



Solution

The output waveform is exactly opposite to the input (inverted), as shown in Figure 3–5, which is the basic timing diagram. The active or asserted output state is **0**.

►**FIGURE 3-5**



Related Problem*

If the inverter is shown with the negative indicator (bubble) on the input instead of the output, how is the timing diagram affected?

*Answers are at the end of the chapter.

Logic Expression for an Inverter

Boolean algebra uses variables and operators to describe a logic circuit.

In **Boolean algebra,** which is the mathematics of logic circuits and will be covered thoroughly in Chapter 4, a variable is generally designated by one or two letters although there can be more. Letters near the beginning of the alphabet usually designate inputs, while letters near the end of the alphabet usually designate outputs. The **complement** of a variable is designated by a bar over the letter. A variable can take on a value of either 1 or 0. If a given variable is 1, its complement is 0 and vice versa.

The operation of an inverter (NOT circuit) can be expressed as follows: If the input variable is called A and the output variable is called X, then

$$X = \overline{A}$$

This expression states that the output is the complement of the input, so if $A = 0$, then $X = 1$, and if $A = 1$, then $X = 0$. Figure 3–6 illustrates this. The complemented variable $\overline{A}$ can be read as "A bar" or "not A."

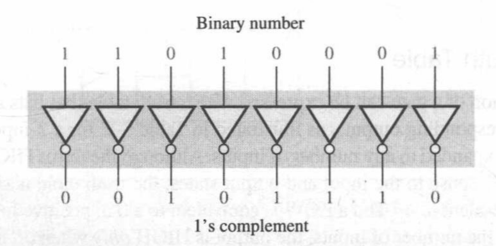
A — [inverter] — $X = \overline{A}$

▲ **FIGURE 3-6**

The inverter complements an input variable.

An Application

Figure 3–7 shows a circuit for producing the 1's complement of an 8-bit binary number. The bits of the binary number are applied to the inverter inputs and the 1's complement of the number appears on the outputs.



► **FIGURE 3-7**

Example of a 1's complement circuit using inverters.

SECTION 3-1 CHECKUP

Answers are at the end of the chapter.

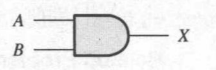
1. When a 1 is on the input of an inverter, what is the output?
2. An active-HIGH pulse (HIGH level when asserted, LOW level when not) is required on an inverter input.
   (a) Draw the appropriate logic symbol, using the distinctive shape and the negation indicator, for the inverter in this application.
   (b) Describe the output when a positive-going pulse is applied to the input of an inverter.

3-2 The AND Gate

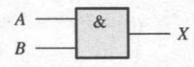
InfoNote

Logic gates are one of the fundamental building blocks of digital systems. Most of the functions in a computer, with the exception of certain types of memory, are implemented with logic gates used on a very large scale. For example, a microprocessor, which is the main part of a computer, is made up of hundreds of thousands or even millions of logic gates.

The term *gate* was introduced in Chapter 1 and is used to describe a circuit that performs a basic logic operation. The AND gate is composed of two or more inputs and a single output, as indicated by the standard logic symbols shown in Figure 3–8. Inputs are on the left, and the output is on the right in each symbol. Gates with two inputs are shown; however, an AND gate can have any number of inputs greater than one. Although examples of both distinctive shape symbols and rectangular outline symbols are shown, the distinctive shape symbol, shown in part (a), is used predominantly in this book.



(a) Distinctive shape



(b) Rectangular outline with the AND (&) qualifying symbol

► **FIGURE 3-8**

Standard logic symbols for the AND gate showing two inputs (ANSI/IEEE Std. 91-1984/Std. 91a-1991).

Operation of an AND Gate

An **AND gate** produces a HIGH output *only* when *all* of the inputs are HIGH. When any of the inputs is LOW, the output is LOW. Therefore, the basic purpose of an AND gate is to determine when certain conditions are simultaneously true, as indicated by HIGH levels on all of its inputs, and to produce a HIGH on its output to indicate that all these conditions are true. The inputs of the 2-input AND gate in Figure 3–8 are labeled *A* and *B*, and the output is labeled *X*. The gate operation can be stated as follows:

An AND gate can have more than two inputs.

For a 2-input AND gate, output *X* is HIGH only when inputs *A* and *B* are HIGH; *X* is LOW when either *A* or *B* is LOW, or when both *A* and *B* are LOW.

Figure 3–9 illustrates a 2-input AND gate with all four possibilities of input combinations and the resulting output for each.

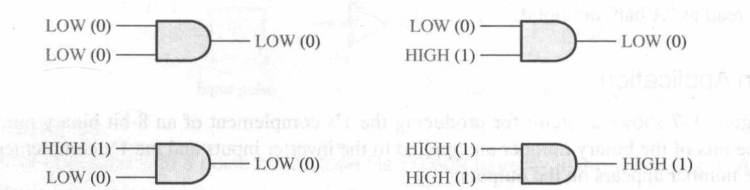


FIGURE 3-9

All possible logic levels for a 2-input AND gate. Open file F03-09 to verify AND gate operation.

MultiSim

AND Gate Truth Table

The logical operation of a gate can be expressed with a truth table that lists all input combinations with the corresponding outputs, as illustrated in Table 3–2 for a 2-input AND gate. The truth table can be expanded to any number of inputs. Although the terms HIGH and LOW tend to give a "physical" sense to the input and output states, the truth table is shown with 1s and 0s; a HIGH is equivalent to a 1 and a LOW is equivalent to a 0 in positive logic. For any AND gate, regardless of the number of inputs, the output is HIGH *only* when *all* inputs are HIGH.

For an AND gate, all HIGH inputs produce a HIGH output.

The total number of possible combinations of binary inputs to a gate is determined by the following formula:

$$N = 2^n \qquad \text{Equation 3–1}$$

where *N* is the number of possible input combinations and *n* is the number of input variables. To illustrate,

For two input variables: $N = 2^2 = 4$ combinations
For three input variables: $N = 2^3 = 8$ combinations
For four input variables: $N = 2^4 = 16$ combinations

You can determine the number of input bit combinations for gates with any number of inputs by using Equation 3–1.

TABLE 3-2

Truth table for a 2-input AND gate.

Inputs		Output
A	*B*	*X*
0	0	0
0	1	0
1	0	0
1	1	1

1 = HIGH, 0 = LOW

EXAMPLE 3-2

(a) Develop the truth table for a 3-input AND gate.

(b) Determine the total number of possible input combinations for a 4-input AND gate.

Inputs			Output
A	*B*	*C*	*X*
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

TABLE 3-3

Solution

(a) There are eight possible input combinations ($2^3 = 8$) for a 3-input AND gate. The input side of the truth table (Table 3–3) shows all eight combinations of three bits. The output side is all 0s except when all three input bits are 1s.

(b) $N = 2^4 = \mathbf{16}$. There are 16 possible combinations of input bits for a 4-input AND gate.

Related Problem

Develop the truth table for a 4-input AND gate.

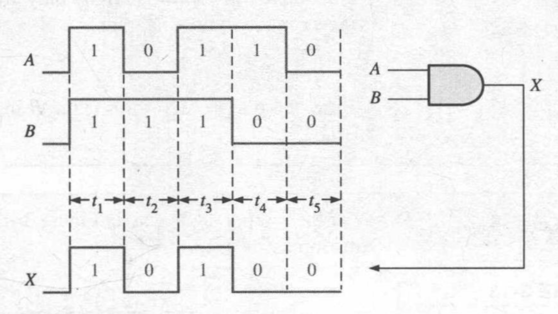
AND Gate Operation with Waveform Inputs

In most applications, the inputs to a gate are not stationary levels but are voltage waveforms that change frequently between HIGH and LOW logic levels. Now let's look at the operation of AND gates with pulse waveform inputs, keeping in mind that an AND gate obeys the truth table operation regardless of whether its inputs are constant levels or levels that change back and forth.

Let's examine the waveform operation of an AND gate by looking at the inputs with respect to each other in order to determine the output level at any given time. In Figure 3–10, inputs A and B are both HIGH (1) during the time interval, t_1, making output X HIGH (1) during this interval. During time interval t_2, input A is LOW (0) and input B is HIGH (1), so the output is LOW (0). During time interval t_3, both inputs are HIGH (1) again, and therefore the output is HIGH (1). During time interval t_4, input A is HIGH (1) and input B is LOW (0), resulting in a LOW (0) output. Finally, during time interval t_5, input A is LOW (0), input B is LOW (0), and the output is therefore LOW (0). As you know, a diagram of input and output waveforms showing time relationships is called a *timing diagram*.

FIGURE 3-10

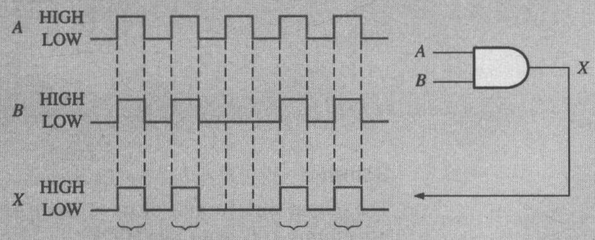
Example of AND gate operation with a timing diagram showing input and output relationships.



EXAMPLE 3-3

If two waveforms, A and B, are applied to the AND gate inputs as in Figure 3–11, what is the resulting output waveform?

FIGURE 3-11



A and *B* are both HIGH during these four time intervals; therefore, *X* is HIGH.

Solution

The output waveform X is HIGH only when both A and B waveforms are HIGH as shown in the timing diagram in Figure 3–11.

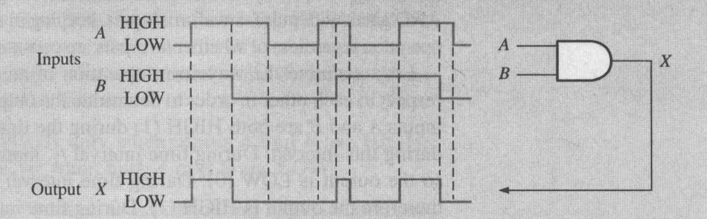
Related Problem

Determine the output waveform and show a timing diagram if the second and fourth pulses in waveform A of Figure 3–11 are replaced by LOW levels.

Remember, when analyzing the waveform operation of logic gates, it is important to pay careful attention to the time relationships of all the inputs with respect to each other and to the output.

EXAMPLE 3-4

For the two input waveforms, *A* and *B*, in Figure 3–12, show the output waveform with its proper relation to the inputs.



▶ FIGURE 3-12

Solution

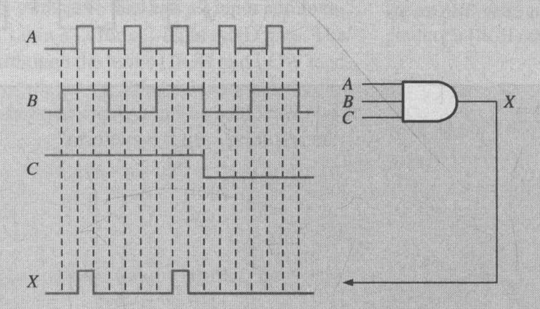
The output waveform is HIGH only when both of the input waveforms are HIGH as shown in the timing diagram.

Related Problem

Show the output waveform if the *B* input to the AND gate in Figure 3–12 is always HIGH.

EXAMPLE 3-5

For the 3-input AND gate in Figure 3–13, determine the output waveform in relation to the inputs.



▶ FIGURE 3-13

Solution

The output waveform *X* of the 3-input AND gate is HIGH only when all three input waveforms *A*, *B*, and *C* are HIGH.

Related Problem

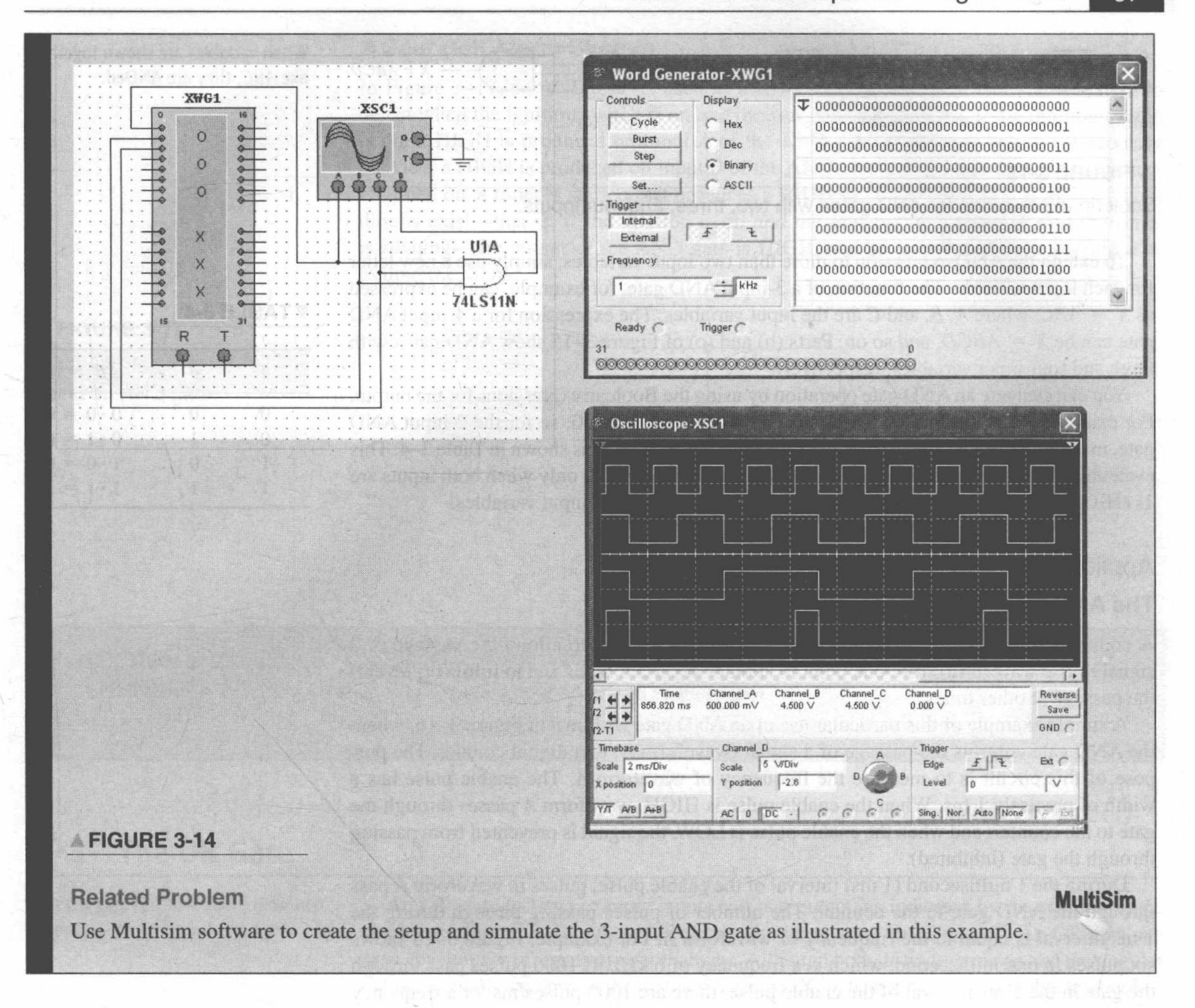
What is the output waveform of the AND gate in Figure 3–13 if the *C* input is always HIGH?

EXAMPLE 3-6

Use Multisim to simulate a 3-input AND gate with input waveforms that cycle through binary numbers 0 through 9.

Solution

Use the Multisim word generator in the up counter mode to provide the combination of waveforms representing the binary sequence, as shown in Figure 3–14. The first three waveforms on the oscilloscope display are the inputs, and the bottom waveform is the output.



▲ **FIGURE 3-14**

Related Problem **MultiSim**

Use Multisim software to create the setup and simulate the 3-input AND gate as illustrated in this example.

Logic Expressions for an AND Gate

The logical AND function of two variables is represented mathematically either by placing a dot between the two variables, as $A \cdot B$, or by simply writing the adjacent letters without the dot, as AB. We will normally use the latter notation.

Boolean multiplication follows the same basic rules governing binary multiplication, which were discussed in Chapter 2 and are as follows:

$$0 \cdot 0 = 0$$
$$0 \cdot 1 = 0$$
$$1 \cdot 0 = 0$$
$$1 \cdot 1 = 1$$

Boolean multiplication is the same as the AND function.

The operation of a 2-input AND gate can be expressed in equation form as follows: If one input variable is A, if the other input variable is B, and if the output variable is X, then the Boolean expression is

$$X = AB$$

Figure 3–15(a) shows the AND gate logic symbol with two input variables and the output variable indicated.

InfoNote

Processors can utilize all of the basic logic operations when it is necessary to selectively manipulate certain bits in one or more bytes of data. Selective bit manipulations are done with a *mask*. For example, to clear (make all 0s) the right four bits in a data byte but keep the left four bits, ANDing the data byte with 11110000 will do the job. Notice that any bit ANDed with zero will be 0 and any bit ANDed with 1 will remain the same. If 10101010 is ANDed with the mask 11110000, the result is 10100000.

A, B — $X = AB$ (a)

A, B, C — $X = ABC$ (b)

A, B, C, D — $X = ABCD$ (c)

▲ **FIGURE 3-15**

Boolean expressions for AND gates with two, three, and four inputs.

When variables are shown together like *ABC*, they are ANDed.

To extend the AND expression to more than two input variables, simply use a new letter for each input variable. The function of a 3-input AND gate, for example, can be expressed as $X = ABC$, where A, B, and C are the input variables. The expression for a 4-input AND gate can be $X = ABCD$, and so on. Parts (b) and (c) of Figure 3–15 show AND gates with three and four input variables, respectively.

You can evaluate an AND gate operation by using the Boolean expressions for the output. For example, each variable on the inputs can be either a 1 or a 0; so for the 2-input AND gate, make substitutions in the equation for the output, $X = AB$, as shown in Table 3–4. This evaluation shows that the output X of an AND gate is a 1 (HIGH) only when both inputs are 1s (HIGHs). A similar analysis can be made for any number of input variables.

▼ **TABLE 3-4**

A	B	$AB = X$
0	0	$0 \cdot 0 = 0$
0	1	$0 \cdot 1 = 0$
1	0	$1 \cdot 0 = 0$
1	1	$1 \cdot 1 = 1$

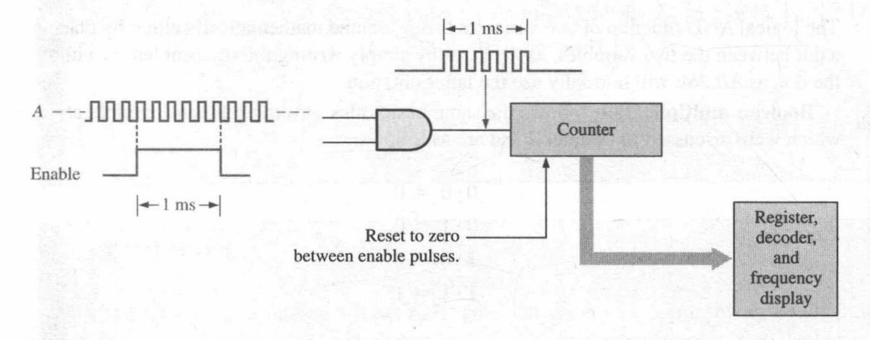
Applications

The AND Gate as an Enable/Inhibit Device

A common application of the AND gate is to **enable** (that is, to allow) the passage of a signal (pulse waveform) from one point to another at certain times and to **inhibit** (prevent) the passage at other times.

A simple example of this particular use of an AND gate is shown in Figure 3–16, where the AND gate controls the passage of a signal (waveform A) to a digital counter. The purpose of this circuit is to measure the frequency of waveform A. The enable pulse has a width of precisely 1 ms. When the enable pulse is HIGH, waveform A passes through the gate to the counter; and when the enable pulse is LOW, the signal is prevented from passing through the gate (inhibited).

During the 1 millisecond (1 ms) interval of the enable pulse, pulses in waveform A pass through the AND gate to the counter. The number of pulses passing through during the 1 ms interval is equal to the frequency of waveform A. For example, Figure 3–16 shows six pulses in one millisecond, which is a frequency of 6 kHz. If 1000 pulses pass through the gate in the 1 ms interval of the enable pulse, there are 1000 pulses/ms, or a frequency of 1 MHz.



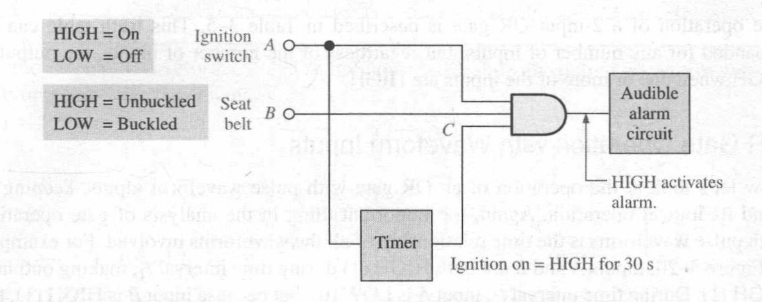
◄ **FIGURE 3-16**

An AND gate performing an enable/inhibit function for a frequency counter.

The counter counts the number of pulses per second and produces a binary output that goes to a decoding and display circuit to produce a readout of the frequency. The enable pulse repeats at certain intervals and a new updated count is made so that if the frequency changes, the new value will be displayed. Between enable pulses, the counter is reset so that it starts at zero each time an enable pulse occurs. The current frequency count is stored in a register so that the display is unaffected by the resetting of the counter.

A Seat Belt Alarm System

In Figure 3–17, an AND gate is used in a simple automobile seat belt alarm system to detect when the ignition switch is on *and* the seat belt is unbuckled. If the ignition switch is on, a HIGH is produced on input *A* of the AND gate. If the seat belt is not properly buckled, a HIGH is produced on input *B* of the AND gate. Also, when the ignition switch is turned on, a timer is started that produces a HIGH on input *C* for 30 s. If all three conditions exist—that is, if the ignition is on *and* the seat belt is unbuckled *and* the timer is running—the output of the AND gate is HIGH, and an audible alarm is energized to remind the driver.



▶ **FIGURE 3-17**
A simple seat belt alarm circuit using an AND gate.

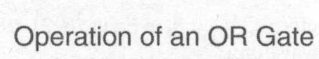
SECTION 3-2 CHECKUP

1. When is the output of an AND gate HIGH?
2. When is the output of an AND gate LOW?
3. Describe the truth table for a 5-input AND gate.

3-3 The OR Gate

An OR gate can have more than two inputs.

An **OR gate** has two or more inputs and one output, as indicated by the standard logic symbols in Figure 3–18, where OR gates with two inputs are illustrated. An OR gate can have any number of inputs greater than one. Although both distinctive shape and rectangular outline symbols are shown, the distinctive shape OR gate symbol is used in this textbook.



(a) Distinctive shape (b) Rectangular outline with the OR (≥ 1) qualifying symbol

▶ **FIGURE 3-18**
Standard logic symbols for the OR gate showing two inputs (ANSI/IEEE Std. 91-1984/Std. 91a-1991).

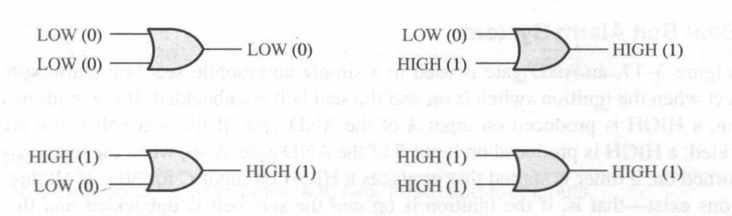
Operation of an OR Gate

For an OR gate, at least one HIGH input produces a HIGH output.

An OR gate produces a HIGH on the output when *any* of the inputs is HIGH. The output is LOW only when all of the inputs are LOW. Therefore, an OR gate determines when one or more of its inputs are HIGH and produces a HIGH on its output to indicate this condition. The inputs of the 2-input OR gate in Figure 3–18 are labeled *A* and *B*, and the output is labeled *X*. The operation of the gate can be stated as follows:

For a 2-input OR gate, output *X* is HIGH when either input *A* or input *B* is HIGH, or when both *A* and *B* are HIGH; *X* is LOW only when both *A* and *B* are LOW.

The HIGH level is the active or asserted output level for the OR gate. Figure 3–19 illustrates the operation for a 2-input OR gate for all four possible input combinations.



◀ **FIGURE 3-19**

All possible logic levels for a 2-input OR gate. Open file F03-19 to verify OR gate operation.

MultiSim

OR Gate Truth Table

The operation of a 2-input OR gate is described in Table 3–5. This truth table can be expanded for any number of inputs; but regardless of the number of inputs, the output is HIGH when one or more of the inputs are HIGH.

Inputs		Output
A	*B*	*X*
0	0	0
0	1	1
1	0	1
1	1	1

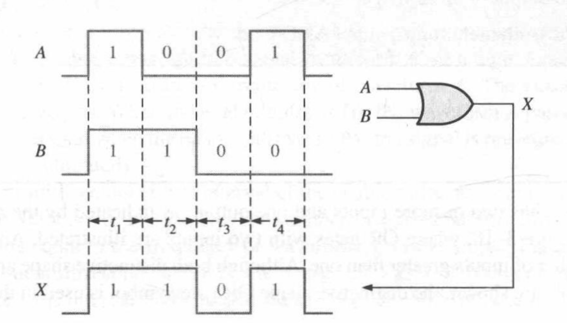
1 = HIGH, 0 = LOW

▲ **TABLE 3-5**

Truth table for a 2-input OR gate.

OR Gate Operation with Waveform Inputs

Now let's look at the operation of an OR gate with pulse waveform inputs, keeping in mind its logical operation. Again, the important thing in the analysis of gate operation with pulse waveforms is the time relationship of all the waveforms involved. For example, in Figure 3–20, inputs *A* and *B* are both HIGH (1) during time interval t_1, making output *X* HIGH (1). During time interval t_2, input *A* is LOW (0), but because input *B* is HIGH (1), the output is HIGH (1). Both inputs are LOW (0) during time interval t_3, so there is a LOW (0) output during this time. During time interval t_4, the output is HIGH (1) because input *A* is HIGH (1).



◀ **FIGURE 3-20**

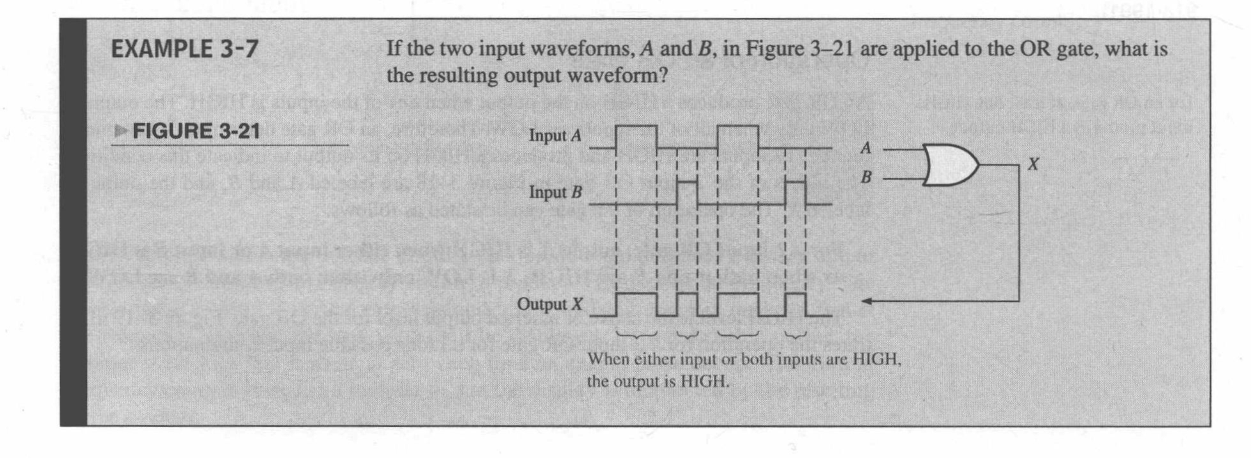
Example of OR gate operation with a timing diagram showing input and output time relationships.

In this illustration, we have applied the truth table operation of the OR gate to each of the time intervals during which the levels are nonchanging. Examples 3–7 through 3–9 further illustrate OR gate operation with waveforms on the inputs.

EXAMPLE 3-7

If the two input waveforms, *A* and *B*, in Figure 3–21 are applied to the OR gate, what is the resulting output waveform?

▶ **FIGURE 3-21**



Solution

The output waveform X of a 2-input OR gate is HIGH when either or both input waveforms are HIGH as shown in the timing diagram. In this case, both input waveforms are never HIGH at the same time.

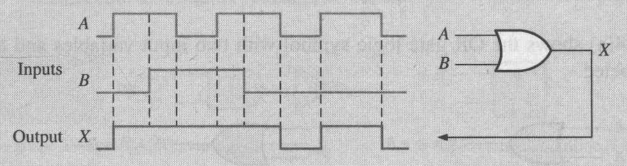
Related Problem

Determine the output waveform and show the timing diagram if input A is changed such that it is HIGH from the beginning of the existing first pulse to the end of the existing second pulse.

EXAMPLE 3-8

For the two input waveforms, A and B, in Figure 3–22, show the output waveform with its proper relation to the inputs.

FIGURE 3-22



Solution

When either or both input waveforms are HIGH, the output is HIGH as shown by the output waveform X in the timing diagram.

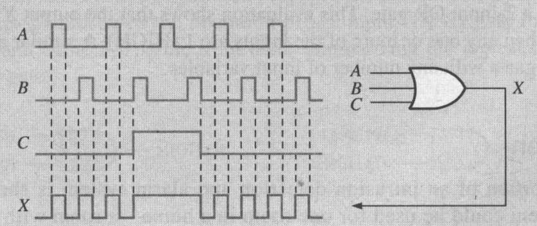
Related Problem

Determine the output waveform and show the timing diagram if the middle pulse of input A is replaced by a LOW level.

EXAMPLE 3-9

For the 3-input OR gate in Figure 3–23, determine the output waveform in proper time relation to the inputs.

FIGURE 3-23



Solution

The output is HIGH when one or more of the input waveforms are HIGH as indicated by the output waveform X in the timing diagram.

Related Problem

Determine the output waveform and show the timing diagram if input C is always LOW.

Logic Expressions for an OR Gate

When variables are separated by +, they are ORed.

The logical OR function of two variables is represented mathematically by a + between the two variables, for example, $A + B$. The plus sign is read as "OR."

Addition in Boolean algebra involves variables whose values are either binary 1 or binary 0. The basic rules for **Boolean addition** are as follows:

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 1$$

Boolean addition is the same as the OR function.

Notice that Boolean addition differs from binary addition in the case where two 1s are added. There is no carry in Boolean addition.

The operation of a 2-input OR gate can be expressed as follows: If one input variable is A, if the other input variable is B, and if the output variable is X, then the Boolean expression is

$$X = A + B$$

Figure 3–24(a) shows the OR gate logic symbol with two input variables and the output variable labeled.

(a) A, B → $X = A + B$	(b) A, B, C → $X = A + B + C$	(c) A, B, C, D → $X = A + B + C + D$

▲ **FIGURE 3-24**
Boolean expressions for OR gates with two, three, and four inputs.

To extend the OR expression to more than two input variables, a new letter is used for each additional variable. For instance, the function of a 3-input OR gate can be expressed as $X = A + B + C$. The expression for a 4-input OR gate can be written as $X = A + B + C + D$, and so on. Parts (b) and (c) of Figure 3–24 show OR gates with three and four input variables, respectively.

OR gate operation can be evaluated by using the Boolean expressions for the output X by substituting all possible combinations of 1 and 0 values for the input variables, as shown in Table 3–6 for a 2-input OR gate. This evaluation shows that the output X of an OR gate is a 1 (HIGH) when any one or more of the inputs are 1 (HIGH). A similar analysis can be extended to OR gates with any number of input variables.

InfoNote

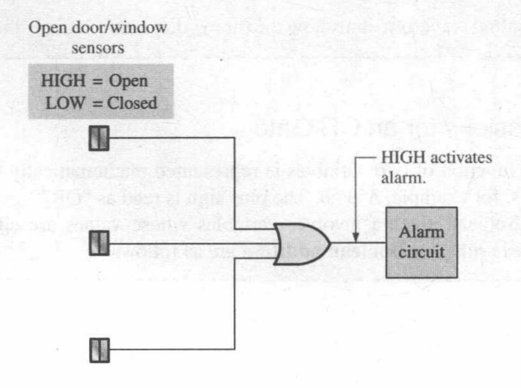
A mask operation that is used in computer programming to selectively make certain bits in a data byte equal to 1 (called setting) while not affecting any other bit is done with the OR operation. A mask is used that contains a 1 in any position where a data bit is to be set. For example, if you want to force the sign bit in an 8-bit signed number to equal 1, but leave all other bits unchanged, you can OR the data byte with the mask 10000000.

▼ **TABLE 3-6**

A	B	$A + B = X$
0	0	$0 + 0 = 0$
0	1	$0 + 1 = 1$
1	0	$1 + 0 = 1$
1	1	$1 + 1 = 1$

An Application

A simplified portion of an intrusion detection and alarm system is shown in Figure 3–25. This system could be used for one room in a home—a room with two windows and a door. The sensors are magnetic switches that produce a HIGH output when open and a LOW output when closed. As long as the windows and the door are secured, the switches are closed and all three of the OR gate inputs are LOW. When one of the windows or the door is opened, a HIGH is produced on that input to the OR gate and the gate output goes HIGH. It then activates and latches an alarm circuit to warn of the intrusion.



◄ **FIGURE 3-25**
A simplified intrusion detection system using an OR gate.

SECTION 3-3 CHECKUP

1. When is the output of an OR gate HIGH?
2. When is the output of an OR gate LOW?
3. Describe the truth table for a 3-input OR gate.

3-4 The NAND Gate

The NAND gate is the same as the AND gate except the output is inverted.

The term *NAND* is a contraction of NOT-AND and implies an AND function with a complemented (inverted) output. The standard logic symbol for a 2-input NAND gate and its equivalency to an AND gate followed by an inverter are shown in Figure 3–26(a), where the symbol ≡ means equivalent to. A rectangular outline symbol is shown in part (b).

▶ **FIGURE 3-26**

Standard NAND gate logic symbols (ANSI/IEEE Std. 91-1984/Std. 91a-1991).

(a) Distinctive shape, 2-input NAND gate and its NOT/AND equivalent

(b) Rectangular outline, 2-input NAND gate with polarity indicator

Operation of a NAND Gate

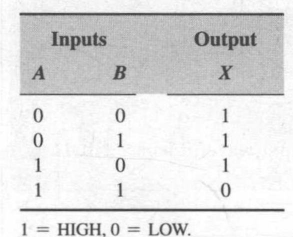
A **NAND gate** produces a LOW output only when all the inputs are HIGH. When any of the inputs is LOW, the output will be HIGH. For the specific case of a 2-input NAND gate, as shown in Figure 3–26 with the inputs labeled *A* and *B* and the output labeled *X*, the operation can be stated as follows:

For a 2-input NAND gate, output *X* is LOW only when inputs *A* and *B* are HIGH; *X* is HIGH when either *A* or *B* is LOW, or when both *A* and *B* are LOW.

This operation is opposite that of the AND in terms of the output level. In a NAND gate, the LOW level (0) is the active or asserted output level, as indicated by the bubble on the output. Figure 3–27 illustrates the operation of a 2-input NAND gate for all four input combinations, and Table 3–7 is the truth table summarizing the logical operation of the 2-input NAND gate.

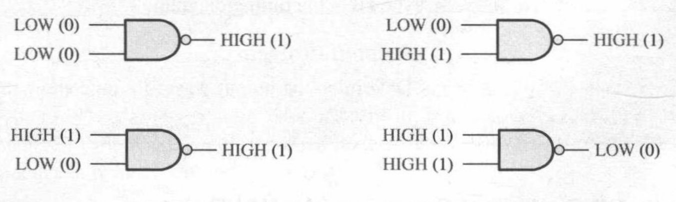
▼ **TABLE 3-7**

Truth table for a 2-input NAND gate.



Inputs		Output
A	*B*	*X*
0	0	1
0	1	1
1	0	1
1	1	0

1 = HIGH, 0 = LOW.



MultiSim

▲ **FIGURE 3-27**

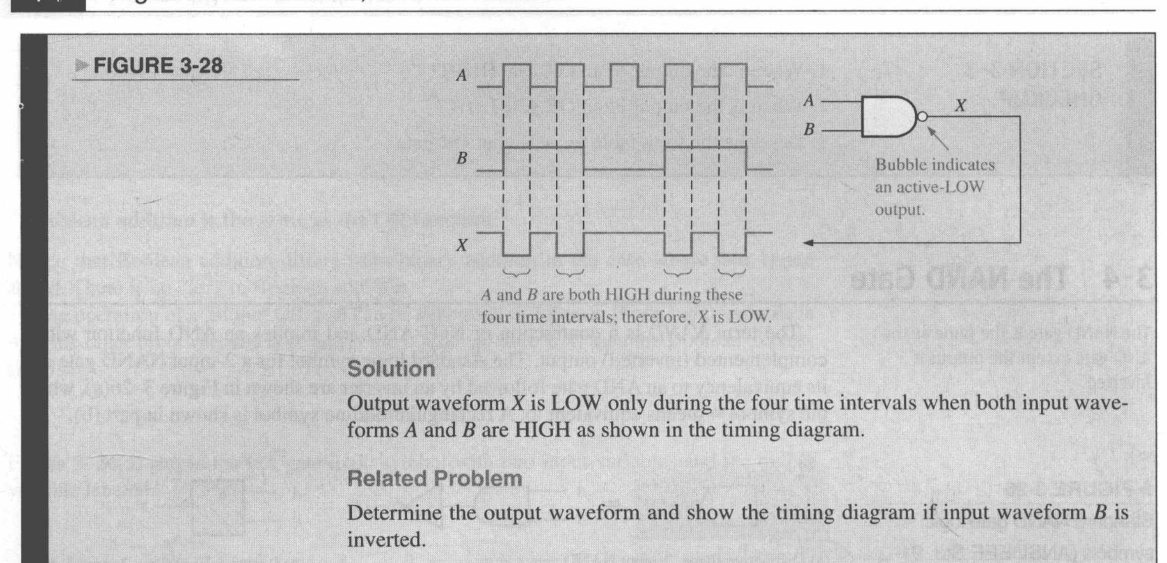
Operation of a 2-input NAND gate. Open file F03-27 to verify NAND gate operation.

NAND Gate Operation with Waveform Inputs

Now let's look at the pulse waveform operation of a NAND gate. Remember from the truth table that the only time a LOW output occurs is when all of the inputs are HIGH.

EXAMPLE 3-10

If the two waveforms *A* and *B* shown in Figure 3–28 are applied to the NAND gate inputs, determine the resulting output waveform.



▶ FIGURE 3-28

Solution

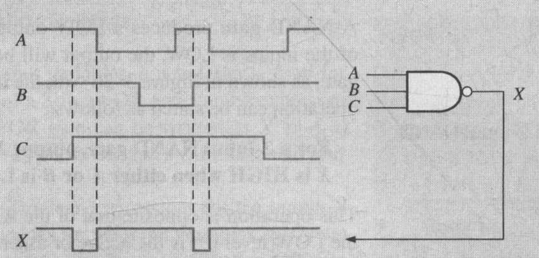
Output waveform X is LOW only during the four time intervals when both input waveforms A and B are HIGH as shown in the timing diagram.

Related Problem

Determine the output waveform and show the timing diagram if input waveform B is inverted.

EXAMPLE 3-11

Show the output waveform for the 3-input NAND gate in Figure 3–29 with its proper time relationship to the inputs.



▶ FIGURE 3-29

Solution

The output waveform X is LOW only when all three input waveforms are HIGH as shown in the timing diagram.

Related Problem

Determine the output waveform and show the timing diagram if input waveform A is inverted.

Negative-OR Equivalent Operation of a NAND Gate

Inherent in a NAND gate's operation is the fact that one or more LOW inputs produce a HIGH output. Table 3–7 shows that output X is HIGH (1) when any of the inputs, A and B, is LOW (0). From this viewpoint, a NAND gate can be used for an OR operation that requires one or more LOW inputs to produce a HIGH output. This aspect of NAND operation is referred to as **negative-OR**. The term *negative* in this context means that the inputs are defined to be in the active or asserted state when LOW.

For a 2-input NAND gate performing a negative-OR operation, output X is HIGH when either input A or input B is LOW, or when both A and B are LOW.

When a NAND gate is used to detect one or more LOWs on its inputs rather than all HIGHs, it is performing the negative-OR operation and is represented by the standard logic symbol shown in Figure 3–30. Although the two symbols in Figure 3–30 represent the same physical gate, they serve to define its role or mode of operation in a particular application, as illustrated by Examples 3–12 and 3–13.

NAND ≡ Negative-OR

▲ FIGURE 3-30

ANSI/IEEE standard symbols representing the two equivalent operations of a NAND gate.

EXAMPLE 3-12

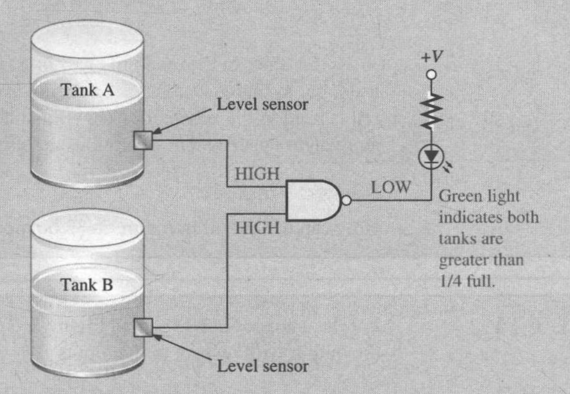
Two tanks store certain liquid chemicals that are required in a manufacturing process. Each tank has a sensor that detects when the chemical level drops to 25% of full. The sensors produce a HIGH level of 5 V when the tanks are more than one-quarter full. When the volume of chemical in a tank drops to one-quarter full, the sensor puts out a LOW level of 0 V.

It is required that a single green light-emitting diode (LED) on an indicator panel show when both tanks are more than one-quarter full. Show how a NAND gate can be used to implement this function.

Solution

Figure 3–31 shows a NAND gate with its two inputs connected to the tank level sensors and its output connected to the indicator panel. The operation can be stated as follows: If tank *A and* tank *B* are above one-quarter full, the LED is on.

▶ **FIGURE 3-31**



As long as both sensor outputs are HIGH (5 V), indicating that both tanks are more than one-quarter full, the NAND gate output is LOW (0 V). The green LED circuit is connected so that a LOW voltage turns it on. The resistor limits the LED current.

Related Problem

How can the circuit of Figure 3–31 be modified to monitor the levels in three tanks rather than two?

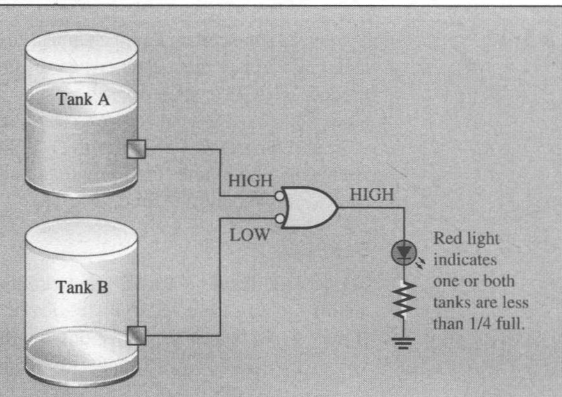
EXAMPLE 3-13

For the process described in Example 3–12 it has been decided to have a red LED display come on when at least one of the tanks falls to the quarter-full level rather than have the green LED display indicate when both are above one quarter. Show how this requirement can be implemented.

Solution

Figure 3–32 shows a NAND gate operating as a negative-OR gate to detect the occurrence of at least one LOW on its inputs. A sensor puts out a LOW voltage if the volume in its tank goes to one-quarter full or less. When this happens, the gate output goes HIGH. The red LED circuit in the panel is connected so that a HIGH voltage turns it on. The operation can be stated as follows: If tank *A or* tank *B or* both are below one-quarter full, the LED is on.

FIGURE 3-32



Notice that, in this example and in Example 3–12, the same 2-input NAND gate is used, but in this example it is operating as a negative-OR gate and a different gate symbol is used in the schematic. This illustrates the different way in which the NAND and equivalent negative-OR operations are used.

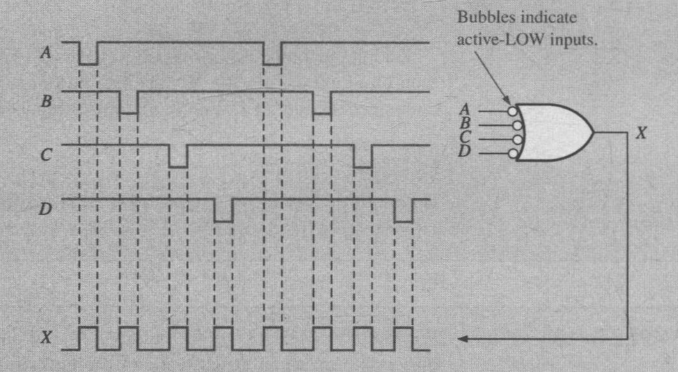
Related Problem

How can the circuit in Figure 3–32 be modified to monitor four tanks rather than two?

EXAMPLE 3-14

For the 4-input NAND gate in Figure 3–33, operating as a negative-OR gate, determine the output with respect to the inputs.

FIGURE 3-33



Solution

The output waveform X is HIGH any time an input waveform is LOW as shown in the timing diagram.

Related Problem

Determine the output waveform if input waveform A is inverted before it is applied to the gate.

Logic Expressions for a NAND Gate

The Boolean expression for the output of a 2-input NAND gate is

$$X = \overline{AB}$$

A bar over a variable or variables indicates an inversion.

▼TABLE 3-8

A	B	$\overline{AB} = X$
0	0	$\overline{0 \cdot 0} = \overline{0} = 1$
0	1	$\overline{0 \cdot 1} = \overline{0} = 1$
1	0	$\overline{1 \cdot 0} = \overline{0} = 1$
1	1	$\overline{1 \cdot 1} = \overline{1} = 0$

This expression says that the two input variables, A and B, are first ANDed and then complemented, as indicated by the bar over the AND expression. This is a description in equation form of the operation of a NAND gate with two inputs. Evaluating this expression for all possible values of the two input variables, you get the results shown in Table 3–8.

Once an expression is determined for a given logic function, that function can be evaluated for all possible values of the variables. The evaluation tells you exactly what the output of the logic circuit is for each of the input conditions, and it therefore gives you a complete description of the circuit's logic operation. The NAND expression can be extended to more than two input variables by including additional letters to represent the other variables.

SECTION 3-4 CHECKUP

1. When is the output of a NAND gate LOW?
2. When is the output of a NAND gate HIGH?
3. Describe the functional differences between a NAND gate and a negative-OR gate. Do they both have the same truth table?
4. Write the output expression for a NAND gate with inputs A, B, and C.

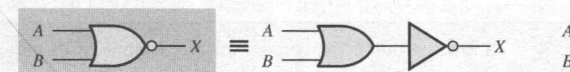
3-5 The NOR Gate

The NOR is the same as the OR except the output is inverted.

The term *NOR* is a contraction of NOT-OR and implies an OR function with an inverted (complemented) output. The standard logic symbol for a 2-input NOR gate and its equivalent OR gate followed by an inverter are shown in Figure 3–34(a). A rectangular outline symbol is shown in part (b).

▶FIGURE 3-34

Standard NOR gate logic symbols (ANSI/IEEE Std. 91-1984/Std. 91a-1991).



(a) Distinctive shape, 2-input NOR gate and its NOT/OR equivalent

(b) Rectangular outline, 2-input NOR gate with polarity indicator

Operation of a NOR Gate

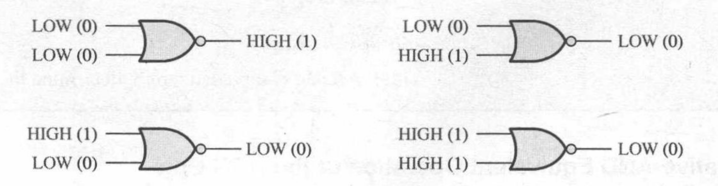
A **NOR gate** produces a LOW output when *any* of its inputs is HIGH. Only when all of its inputs are LOW is the output HIGH. For the specific case of a 2-input NOR gate, as shown in Figure 3–34 with the inputs labeled A and B and the output labeled X, the operation can be stated as follows:

For a 2-input NOR gate, output X is LOW when either input A or input B is HIGH, or when both A and B are HIGH; X is HIGH only when both A and B are LOW.

▶FIGURE 3-35

Operation of a 2-input NOR gate. Open file F03-35 to verify NOR gate operation.

MultiSim



This operation results in an output level opposite that of the OR gate. In a NOR gate, the LOW output is the active or asserted output level as indicated by the bubble on the output. Figure 3–35 illustrates the operation of a 2-input NOR gate for all four possible input combinations, and Table 3–9 is the truth table for a 2-input NOR gate.

Inputs		Output
A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

1 = HIGH, 0 = LOW.

NOR Gate Operation with Waveform Inputs

The next two examples illustrate the operation of a NOR gate with pulse waveform inputs. Again, as with the other types of gates, we will simply follow the truth table operation to determine the output waveforms in the proper time relationship to the inputs.

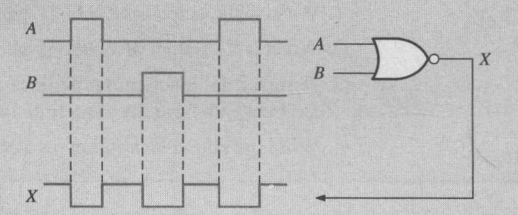
◀ **TABLE 3-9**

Truth table for a 2-input NOR gate.

EXAMPLE 3-15

If the two waveforms shown in Figure 3–36 are applied to a NOR gate, what is the resulting output waveform?

▶ **FIGURE 3-36**



Solution

Whenever any input of the NOR gate is HIGH, the output is LOW as shown by the output waveform *X* in the timing diagram.

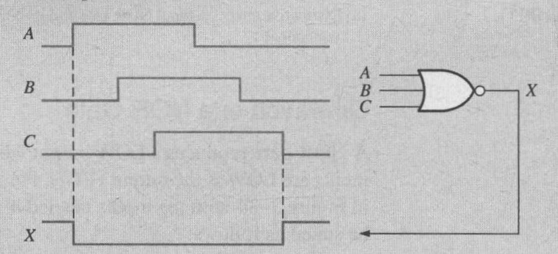
Related Problem

Invert input *B* and determine the output waveform in relation to the inputs.

EXAMPLE 3-16

Show the output waveform for the 3-input NOR gate in Figure 3–37 with the proper time relation to the inputs.

▶ **FIGURE 3-37**



Solution

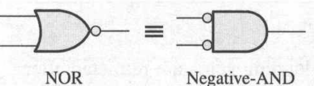
The output *X* is LOW when any input is HIGH as shown by the output waveform *X* in the timing diagram.

Related Problem

With the *B* and *C* inputs inverted, determine the output and show the timing diagram.

Negative-AND Equivalent Operation of the NOR Gate

A NOR gate, like the NAND, has another aspect of its operation that is inherent in the way it logically functions. Table 3–9 shows that a HIGH is produced on the gate output only when all of the inputs are LOW. From this viewpoint, a NOR gate can be used for an AND operation that requires all LOW inputs to produce a HIGH output. This aspect of NOR operation is called **negative-AND**. The term *negative* in this context means that the inputs are defined to be in the active or asserted state when LOW.



▲ **FIGURE 3-38**
Standard symbols representing the two equivalent operations of a NOR gate.

For a 2-input NOR gate performing a negative-AND operation, output *X* is HIGH only when both inputs *A* and *B* are LOW.

When a NOR gate is used to detect all LOWs on its inputs rather than one or more HIGHs, it is performing the negative-AND operation and is represented by the standard symbol in Figure 3–38. Remember that the two symbols in Figure 3–38 represent the same physical gate and serve only to distinguish between the two modes of its operation. The following three examples illustrate this.

EXAMPLE 3-17

A device is needed to indicate when two LOW levels occur simultaneously on its inputs and to produce a HIGH output as an indication. Specify the device.

Solution

A 2-input NOR gate operating as a negative-AND gate is required to produce a HIGH output when both inputs are LOW, as shown in Figure 3–39.

▶ **FIGURE 3-39**

Related Problem

A device is needed to indicate when one or two HIGH levels occur on its inputs and to produce a LOW output as an indication. Specify the device.

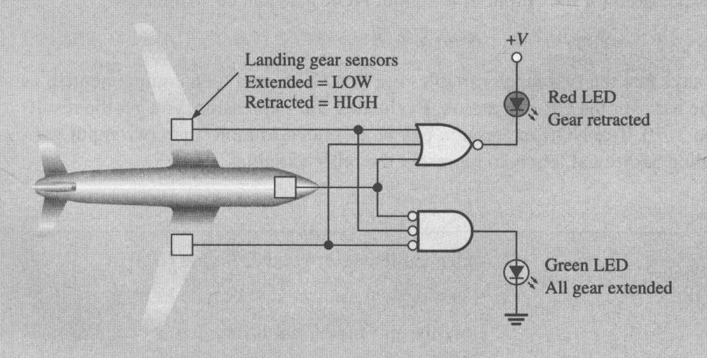
EXAMPLE 3-18

As part of an aircraft's functional monitoring system, a circuit is required to indicate the status of the landing gears prior to landing. A green LED display turns on if all three gears are properly extended when the "gear down" switch has been activated in preparation for landing. A red LED display turns on if any of the gears fail to extend properly prior to landing. When a landing gear is extended, its sensor produces a LOW voltage. When a landing gear is retracted, its sensor produces a HIGH voltage. Implement a circuit to meet this requirement.

Solution

Power is applied to the circuit only when the "gear down" switch is activated. Use a NOR gate for each of the two requirements as shown in Figure 3–40. One NOR gate operates as a negative-AND to detect a LOW from each of the three landing gear sensors. When all three of the gate inputs are LOW, the three landing gears are properly extended and the resulting HIGH output from the negative-AND gate turns on the green LED display. The other NOR gate operates as a NOR to detect if one or more of the landing gears remain retracted when the "gear down" switch is activated. When one or more of the landing gears remain retracted, the resulting HIGH from the sensor is detected by the NOR gate, which produces a LOW output to turn on the red LED warning display.

▶ **FIGURE 3-40**



Related Problem

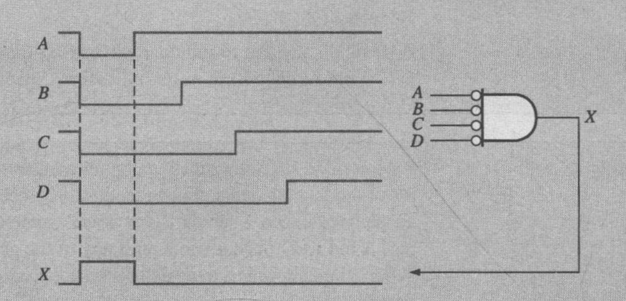
What type of gate should be used to detect if all three landing gears are retracted after takeoff, assuming a LOW output is required to activate an LED display?

When driving a load such as an LED with a logic gate, consult the manufacturer's data sheet for maximum drive capabilities (output current). A regular IC logic gate may not be capable of handling the current required by certain loads such as some LEDs. Logic gates with a buffered output, such as an open-collector (OC) or open-drain (OD) output, are available in many types of IC logic gate configurations. The output current capability of typical IC logic gates is limited to the μA or relatively low mA range. For example, standard TTL can handle output currents up to 16 mA but only when the output is LOW. Most LEDs require currents in the range of about 10 mA to 50 mA.

EXAMPLE 3-19

For the 4-input NOR gate operating as a negative-AND in Figure 3–41, determine the output relative to the inputs.

▶ FIGURE 3-41



Solution

Any time all of the input waveforms are LOW, the output is HIGH as shown by output waveform X in the timing diagram.

Related Problem

Determine the output with input D inverted and show the timing diagram.

Logic Expressions for a NOR Gate

The Boolean expression for the output of a 2-input NOR gate can be written as

$$X = \overline{A + B}$$

This equation says that the two input variables are first ORed and then complemented, as indicated by the bar over the OR expression. Evaluating this expression, you get the results shown in Table 3–10. The NOR expression can be extended to more than two input variables by including additional letters to represent the other variables.

▼ TABLE 3-10

A	B	$\overline{A + B} = X$
0	0	$\overline{0 + 0} = \overline{0} = 1$
0	1	$\overline{0 + 1} = \overline{1} = 0$
1	0	$\overline{1 + 0} = \overline{1} = 0$
1	1	$\overline{1 + 1} = \overline{1} = 0$

SECTION 3-5 CHECKUP

1. When is the output of a NOR gate HIGH?
2. When is the output of a NOR gate LOW?
3. Describe the functional difference between a NOR gate and a negative-AND gate. Do they both have the same truth table?
4. Write the output expression for a 3-input NOR with input variables A, B, and C.

3-6 The Exclusive-OR and Exclusive-NOR Gates

InfoNote

Exclusive-OR gates connected to form an adder circuit allow a processor to perform addition, subtraction, multiplication, and division in its Arithmetic Logic Unit (ALU). An exclusive-OR gate combines basic AND, OR, and NOT logic.

The Exclusive-OR Gate

Standard symbols for an exclusive-OR (XOR for short) gate are shown in Figure 3–42. The XOR gate has only two inputs. The **exclusive-OR gate** performs modulo-2 addition (introduced in Chapter 2). The output of an exclusive-OR gate is HIGH *only* when the two inputs are at opposite logic levels. This operation can be stated as follows with reference to inputs A and B and output X:

A, B → X　　A, B → [=1] → X

(a) Distinctive shape　　(b) Rectangular outline

▲ **FIGURE 3-42**

Standard logic symbols for the exclusive-OR gate.

For an exclusive-OR gate, opposite inputs make the output HIGH.

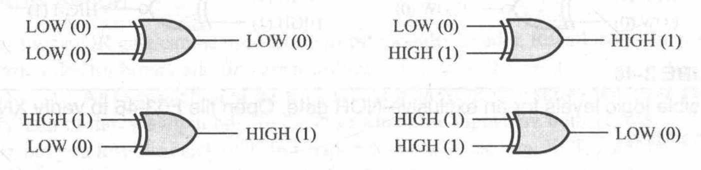
For an exclusive-OR gate, output X is HIGH when input A is LOW and input B is HIGH, or when input A is HIGH and input B is LOW; X is LOW when A and B are both HIGH or both LOW.

▼ **TABLE 3-11**

Truth table for an exclusive-OR gate.

Inputs		Output
A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

The four possible input combinations and the resulting outputs for an XOR gate are illustrated in Figure 3–43. The HIGH level is the active or asserted output level and occurs only when the inputs are at opposite levels. The operation of an XOR gate is summarized in the truth table shown in Table 3–11.



▲ **FIGURE 3-43**

MultiSim

All possible logic levels for an exclusive-OR gate. Open file F03-43 to verify XOR gate operation.

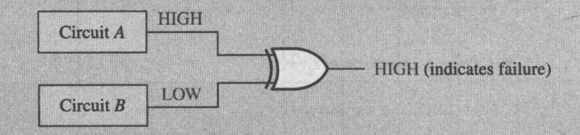
EXAMPLE 3-20

A certain system contains two identical circuits operating in parallel. As long as both are operating properly, the outputs of both circuits are always the same. If one of the circuits fails, the outputs will be at opposite levels at some time. Devise a way to monitor and detect that a failure has occurred in one of the circuits.

Solution

The outputs of the circuits are connected to the inputs of an XOR gate as shown in Figure 3–44. A failure in either one of the circuits produces differing outputs, which cause the XOR inputs to be at opposite levels. This condition produces a HIGH on the output of the XOR gate, indicating a failure in one of the circuits.

▶ **FIGURE 3-44**



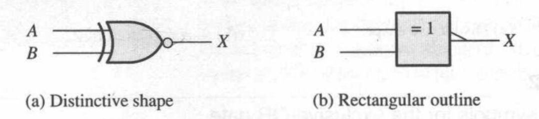
Related Problem

Will the exclusive-OR gate always detect simultaneous failures in both circuits of Figure 3–44? If not, under what condition?

The Exclusive-NOR Gate

Standard symbols for an **exclusive-NOR (XNOR) gate** are shown in Figure 3–45. Like the XOR gate, an XNOR has only two inputs. The bubble on the output of the XNOR symbol indicates that its output is opposite that of the XOR gate. When the two input logic levels are opposite, the output of the exclusive-NOR gate is LOW. The operation can be stated as follows (*A* and *B* are inputs, *X* is the output):

For an exclusive-NOR gate, output *X* is LOW when input *A* is LOW and input *B* is HIGH, or when *A* is HIGH and *B* is LOW; *X* is HIGH when *A* and *B* are both HIGH or both LOW.



▲ **FIGURE 3-45**

Standard logic symbols for the exclusive-NOR gate.

▼ **TABLE 3-12**

Truth table for an exclusive-NOR gate.

Inputs		Output
A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

The four possible input combinations and the resulting outputs for an XNOR gate are shown in Figure 3–46. The operation of an XNOR gate is summarized in Table 3–12. Notice that the output is HIGH when the same level is on both inputs.

LOW (0), LOW (0) → HIGH (1)
LOW (0), HIGH (1) → LOW (0)
HIGH (1), LOW (0) → LOW (0)
HIGH (1), HIGH (1) → HIGH (1)

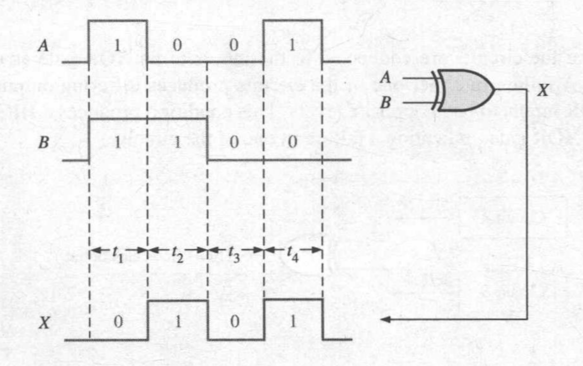
▲ **FIGURE 3-46**

All possible logic levels for an exclusive-NOR gate. Open file F03-46 to verify XNOR gate operation.

MultiSim

Operation with Waveform Inputs

As we have done with the other gates, let's examine the operation of XOR and XNOR gates with pulse waveform inputs. As before, we apply the truth table operation during each distinct time interval of the pulse waveform inputs, as illustrated in Figure 3–47 for an XOR gate. You can see that the input waveforms *A* and *B* are at opposite levels during time intervals t_2 and t_4. Therefore, the output *X* is HIGH during these two times. Since both inputs are at the same level, either both HIGH or both LOW, during time intervals t_1 and t_3, the output is LOW during those times as shown in the timing diagram.



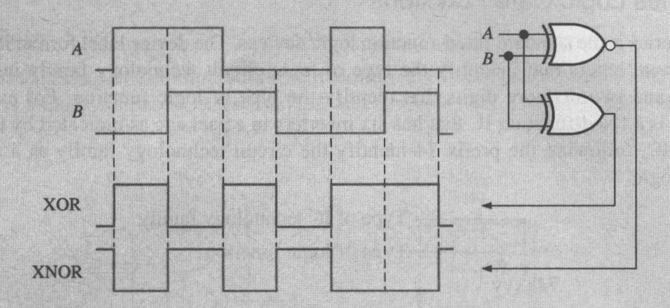
▲ **FIGURE 3-47**

Example of exclusive-OR gate operation with pulse waveform inputs.

EXAMPLE 3-21

Determine the output waveforms for the XOR gate and for the XNOR gate, given the input waveforms, *A* and *B*, in Figure 3–48.

▶ **FIGURE 3-48**



Solution

The output waveforms are shown in Figure 3–48. Notice that the XOR output is HIGH only when both inputs are at opposite levels. Notice that the XNOR output is HIGH only when both inputs are the same.

Related Problem

Determine the output waveforms if the two input waveforms, *A* and *B*, are inverted.

An Application

An exclusive-OR gate can be used as a two-bit modulo-2 adder. Recall from Chapter 2 that the basic rules for binary addition are as follows: $0 + 0 = 0$, $0 + 1 = 1$, $1 + 0 = 1$, and $1 + 1 = 10$. An examination of the truth table for an XOR gate shows that its output is the binary sum of the two input bits. In the case where the inputs are both 1s, the output is the sum 0, but you lose the carry of 1. In Chapter 6 you will see how XOR gates are combined to make complete adding circuits. Table 3–13 illustrates an XOR gate used as a modulo-2 adder. It is used in CRC systems to implement the division process that was described in Chapter 2.

▶ **TABLE 3-13**

An XOR gate used to add two bits.

Input Bits		Output (Sum)
A	*B*	Σ
0	0	0
0	1	1
1	0	1
1	1	0 (without the 1 carry bit)

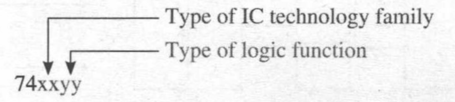
SECTION 3-6 CHECKUP

1. When is the output of an XOR gate HIGH?
2. When is the output of an XNOR gate HIGH?
3. How can you use an XOR gate to detect when two bits are different?

3-7 Fixed-Function Logic Gates

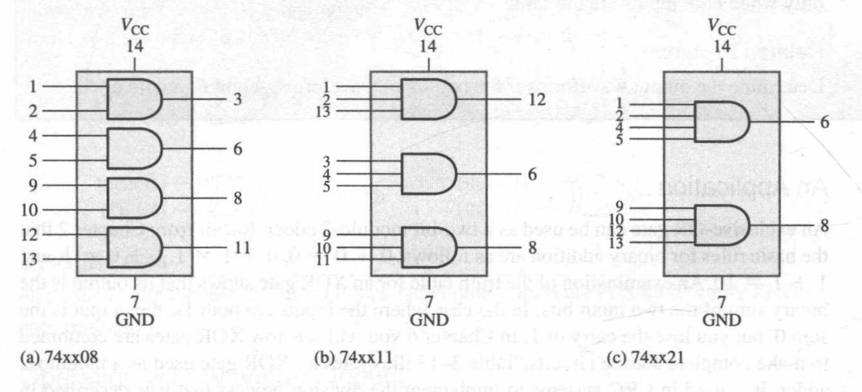
74 Series Logic Gate Functions

The 74 series is the standard fixed-function logic devices. The device label format includes one or more letters that indentify the type of logic circuit technology family in the IC package and two or more digits that identify the type of logic function. For example, 74HC04 is a fixed-function IC that has six inverters in a package as indicated by 04. The letters, HC, following the prefix 74 identify the circuit technology family as a type of CMOS logic.



AND Gate

Figure 3–49 shows three configurations of fixed-function AND gates in the 74 series. The 74xx08 is a quad 2-input AND gate device, the 74xx11 is a triple 3-input AND gate device,

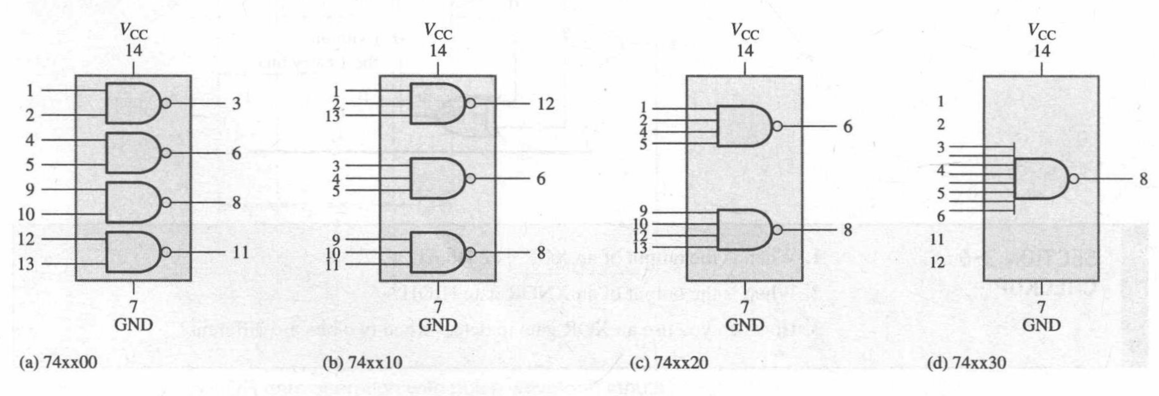


◄ **FIGURE 3-49**
74 series AND gate devices with pin numbers.

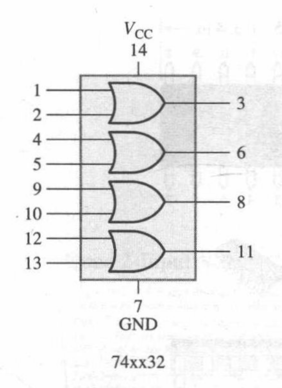
and the 74xx21 is a dual 4-input AND gate device. The label xx can represent any of the integrated circuit technology families such as HC or LS. The numbers on the inputs and outputs are the IC package pin numbers.

NAND Gate

Figure 3–50 shows four configurations of fixed-function NAND gates in the 74 series. The 74xx00 is a quad 2-input NAND gate device, the 74xx10 is a triple 3-input NAND gate device, the 74xx20 is a dual 4-input NAND gate device, and the 74xx30 is a single 8-input NAND gate device.



▲ **FIGURE 3-50**
74 series NAND gate devices with package pin numbers.



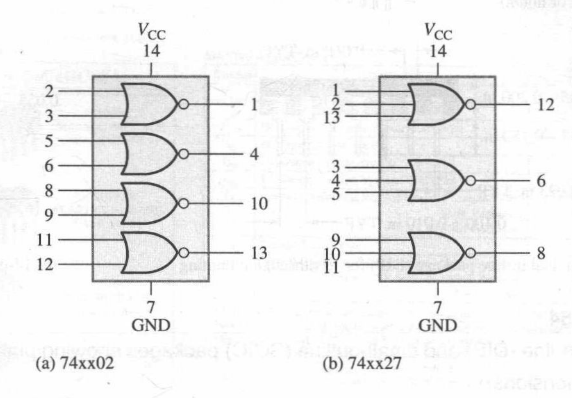
▲ **FIGURE 3-51**
74 series OR gate device.

OR Gate

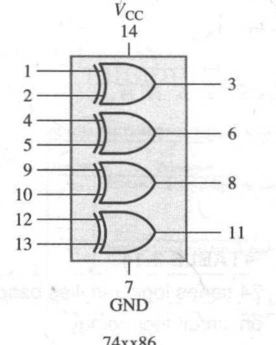
Figure 3–51 shows a fixed-function OR gate in the 74 series. The 74xx32 is a quad 2-input OR gate device.

NOR Gate

Figure 3–52 shows two configurations of fixed-function NOR gates in the 74 series. The 74xx02 is a quad 2-input NOR gate device, and the 74xx27 is a triple 3-input NOR gate device.



▲ **FIGURE 3-52**
74 series NOR gate devices.



▲ **FIGURE 3-53**
74 series XOR gate.

XOR Gate

Figure 3–53 shows a fixed-function XOR (exclusive-OR) gate in the 74 series. The 74xx86 is a quad 2-input XOR gate.

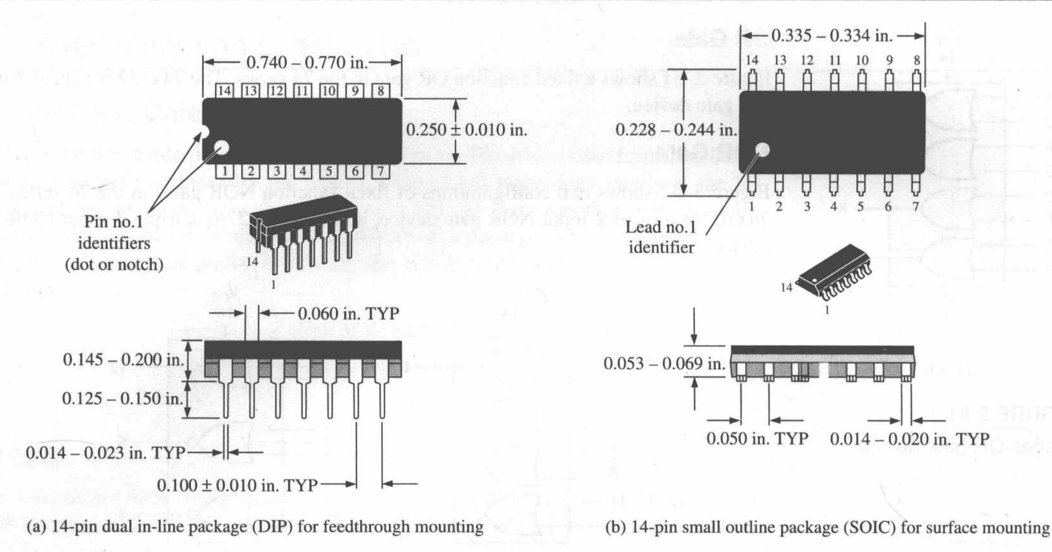
IC Packages

All of the 74 series CMOS are pin-compatible with the same types of devices in bipolar. This means that a CMOS digital IC such as the 74HC00 (quad 2-input NAND), which contains four 2-input NAND gates in one IC package, has the identical package pin numbers for each input and output as does the corresponding bipolar device. Typical IC gate packages, the dual in-line package (DIP) for plug-in or feedthrough mounting and the small-outline integrated circuit (SOIC) package for surface mounting, are shown in Figure 3–54. In some cases, other types of packages are also available. The SOIC package is significantly smaller than the DIP. Packages with a single gate are known as *little logic*. Most logic gate functions are available and are implemented in a CMOS circuit technology. Typically, the gates have only two inputs and have a different designation than multigate devices. For example, the 74xx1G00 is a single 2-input NAND gate.

Handling Precautions for CMOS

CMOS logic is very sensitive to static charge and can be damaged by ESD (electrostatic discharge) if not handled properly as follows:

1. Store and ship in conductive foam.
2. Connect instruments to earth ground.
3. Connect wrist to earth ground through a large series resistor.
4. Do not remove devices from circuit with power on.
5. Do not apply signal voltage when power is off.



(a) 14-pin dual in-line package (DIP) for feedthrough mounting

(b) 14-pin small outline package (SOIC) for surface mounting

▲ **FIGURE 3-54**

Typical dual in-line (DIP) and small-outline (SOIC) packages showing pin numbers and basic dimensions.

74 Series Logic Circuit Families

Although many logic circuit families have become obsolete and some are rapidly on the decline, others are still very active and available. CMOS is the most available and most popular type of logic circuit technology, and the HC (high-speed CMOS) family is the most recommended for new projects. For bipolar, the LS (low-power schottky) family is the most widely used. The HCT, which a variation of the HC family, is compatible with bipolar devices such as LS.

Table 3–14 lists many logic circuit technology families. Because the active status of any given logic family is always in flux, check with a manufacturer, such as Texas Instruments, for information on active/nonactive status and availability for a logic function in a given circuit technology.

◀ **TABLE 3-14**

74 series logic families based on circuit technology.

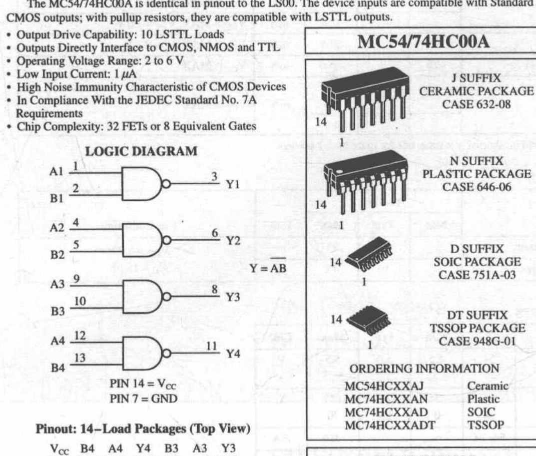
Circuit Type	Description	Circuit Technology
ABT	Advanced BiCMOS	BiCMOS
AC	Advanced CMOS	CMOS
ACT	Bipolar compatible AC	CMOS
AHC	Advanced high-speed CMOS	CMOS
AHCT	Bipolar compatible AHC	CMOS
ALB	Advanced low-voltage BiCMOS	BiCMOS
ALS	Advanced low-power Schottky	Bipolar
ALVC	Advanced low-voltage CMOS	CMOS
AUC	Advanced ultra-low-voltage CMOS	CMOS
AUP	Advanced ultra-low-power CMOS	CMOS
AS	Advanced Schottky	Bipolar
AVC	Advanced very-low-power CMOS	CMOS
BCT	Standard BiCMOS	BiCMOS
F	Fast	Bipolar
FCT	Fast CMOS technology	CMOS
HC	High-speed CMOS	CMOS
HCT	Bipolar compatible HC	CMOS
LS	Low-power Schottky	Bipolar
LV-A	Low-voltage CMOS	CMOS
LV-AT	Bipolar compatible LV-A	CMOS
LVC	Low-voltage CMOS	CMOS
LVT	Low-voltage biCMOS	BiCMOS
S	Schottky	Bipolar

The type of integrated circuit technology has nothing to do with the logic function itself. For example, the 74HC00, 74HCT00, and 74LS00 are all quad 2-input NAND gates with identical package pin configurations. The differences among these three logic devices are in the electrical and performance characteristics such as power consumption, dc supply voltage, switching speed, and input/output voltage levels. CMOS and bipolar circuits are implemented with two different types of transistors. Figures 3–55 and 3–56 show partial data sheets for the 74HC00A quad 2-input NAND gate in CMOS and in bipolar technologies, respectively.

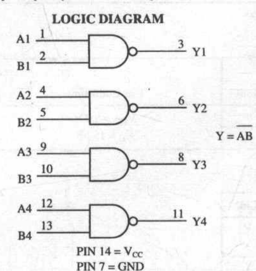
Quad 2-Input NAND Gate High-Performance Silicon–Gate CMOS

The MC54/74HC00A is identical in pinout to the LS00. The device inputs are compatible with Standard CMOS outputs; with pullup resistors, they are compatible with LSTTL outputs.

- Output Drive Capability: 10 LSTTL Loads
- Outputs Directly Interface to CMOS, NMOS and TTL
- Operating Voltage Range: 2 to 6 V
- Low Input Current: 1 μA
- High Noise Immunity Characteristic of CMOS Devices
- In Compliance With the JEDEC Standard No. 7A Requirements
- Chip Complexity: 32 FETs or 8 Equivalent Gates



LOGIC DIAGRAM



ORDERING INFORMATION

MC54HCXXAJ	Ceramic
MC74HCXXAN	Plastic
MC74HCXXAD	SOIC
MC74HCXXADT	TSSOP

Pinout: 14–Load Packages (Top View)

V_{CC} B4 A4 Y4 B3 A3 Y3
14 13 12 11 10 9 8

1 2 3 4 5 6 7
A1 B1 Y1 A2 B2 Y2 GND

FUNCTION TABLE

Inputs		Output
A	B	Y
L	L	H
L	H	H
H	L	H
H	H	L

MAXIMUM RATINGS*

Symbol	Parameter	Value	Unit
V_{CC}	DC Supply Voltage (Referenced to GND)	–0.5 to + 7.0	V
V_{in}	DC Input Voltage (Referenced to GND)	–0.5 to V_{CC} + 0.5	V
V_{out}	DC Output Voltage (Referenced to GND)	–0.5 to V_{CC} + 0.5	V
I_{in}	DC Input Current, per Pin	± 20	mA
I_{out}	DC Output Current, per Pin	± 25	mA
I_{CC}	DC Supply Current, V_{CC} and GND Pins	± 50	mA
P_D	Power Dissipation in Still Air, Plastic or Ceramic DIP†	750	mW
	SOIC Package†	500	
	TSSOP Package†	450	
T_{stg}	Storage Temperature	–65 to + 150	°C
T_L	Lead Temperature, 1 mm from Case for 10 Seconds		°C
	Plastic DIP, SOIC or TSSOP Package	260	
	Ceramic DIP	300	

* Maximum Ratings are those values beyond which damage to the device may occur. Functional operation should be restricted to the Recommended Operating Conditions.

† Derating — Plastic DIP: – 10 mW/°C from 65° to 125° C
Ceramic DIP: – 10 mW/°C from 100° to 125° C
SOIC Package: – 7 mW/°C from 65° to 125° C
TSSOP Package: – 6.1 mW/°C from 65° to 125° C

RECOMMENDED OPERATING CONDITIONS

Symbol	Parameter		in	Max	Unit
V_{CC}	DC Supply Voltage (Referenced to GND)		2.0	6.0	V
V_{in}, V_{out}	DC Input Voltage, Output Voltage (Referenced to GND)		0	V_{CC}	V
T_A	Operating Temperature, All Package Types		-55	+125	°C
t_r, t_f	Input Rise and Fall Time	V_{CC} = 2.0 V	0	1000	ns
		V_{CC} = 4.5 V	0	500	
		V_{CC} = 6.0 V	0	400	

DC CHARACTERISTICS (Voltages Referenced to GND) MC54/74HC00A

Symbol	Parameter	Condition	V_{CC} V	Guaranteed Limit –55 to 25°C	≤85°C	≤125°C	Unit
V_{IH}	Minimum High-Level Input Voltage	V_{out} = 0.1V or V_{CC} – 0.1V, $\lvert I_{out}\rvert \le 20\mu A$	2.0	1.50	1.50	1.50	V
			3.0	2.10	2.10	2.10	
			4.5	3.15	3.15	3.15	
			6.0	4.20	4.20	4.20	
V_{IL}	Maximum Low-Level Input Voltage	V_{out} = 0.1V or V_{CC} – 0.1V, $\lvert I_{out}\rvert \le 20\mu A$	2.0	0.50	0.50	0.50	V
			3.0	0.90	0.90	0.90	
			4.5	1.35	1.35	1.35	
			6.0	1.80	1.80	1.80	
V_{OH}	Minimum High-Level Output Voltage	$V_{in} = V_{IH}$ or V_{IL}, $\lvert I_{out}\rvert \le 20\mu A$	2.0	1.9	1.9	1.9	V
			4.5	4.4	4.4	4.4	
			6.0	5.9	5.9	5.9	
		$V_{in} = V_{IH}$ or V_{IL}, $\lvert I_{out}\rvert \le 2.4mA$	3.0	2.48	2.34	2.20	
		$\lvert I_{out}\rvert \le 4.0mA$	4.5	3.98	3.84	3.70	
		$\lvert I_{out}\rvert \le 5.2mA$	6.0	5.48	5.34	5.20	
V_{OL}	Maximum Low-Level Output Voltage	$V_{in} = V_{IH}$ or V_{IL}, $\lvert I_{out}\rvert \le 20\mu A$	2.0	0.1	0.1	0.1	V
			4.5	0.1	0.1	0.1	
			6.0	0.1	0.1	0.1	
		$V_{in} = V_{IH}$ or V_{IL}, $\lvert I_{out}\rvert \le 2.4mA$	3.0	0.26	0.33	0.40	
		$\lvert I_{out}\rvert \le 4.0mA$	4.5	0.26	0.33	0.40	
		$\lvert I_{out}\rvert \le 5.2mA$	6.0	0.26	0.33	0.40	
I_{in}	Maximum Input Leakage Current	$V_{in} = V_{CC}$ or GND	6.0	±0.1	±1.0	±1.0	μA
I_{CC}	Maximum Quiescent Supply Current (per Package)	$V_{in} = V_{CC}$ or GND, $I_{out} = 0\mu A$	6.0	1.0	10	40	μA

AC CHARACTERISTICS (C_L = 50 pF, Input $t_r = t_f$ = 6 ns)

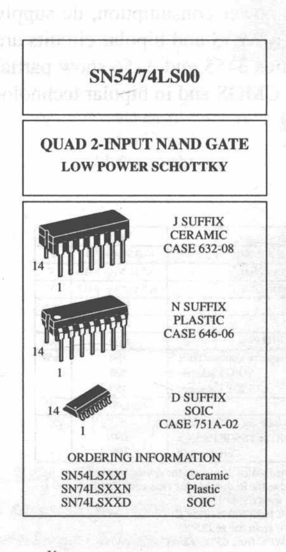
Symbol	Parameter	V_{CC} V	Guaranteed Limit –55 to 25°C	≤85°C	≤125°C	Unit
t_{PLH}, t_{PHL}	Maximum Propagation Delay, Input A or B to Output Y	2.0	75	95	110	ns
		3.0	30	40	55	
		4.5	15	19	22	
		6.0	13	16	19	
t_{TLH}, t_{THL}	Maximum Output Transition Time, Any Output	2.0	75	95	110	ns
		3.0	27	32	36	
		4.5	15	19	22	
		6.0	13	16	19	
C_{in}	Maximum Input Capacitance		10	10	10	pF

		Typical @ 25°C, V_{CC} = 5.0 V, V_{EE} = 0 V	
C_{PD}	Power Dissipation Capacitance (Per Buffer)	22	pF

FIGURE 3-55 CMOS logic. Partial data sheet for a 54/74HC00A quad 2-input NAND gate. The 54 prefix indicates military grade and the 74 prefix indicates commercial grade.

QUAD 2-INPUT NAND GATE

• ESD > 3500 Volts



SN54/74LS00

DC CHARACTERISTICS OVER OPERATING TEMPERATURE RANGE (unless otherwise specified)

Symbol	Parameter		Limits Min	Typ	Max	Unit	Test Conditions
V_{IH}	Input HIGH Voltage		2.0			V	Guaranteed Input HIGH Voltage for All Inputs
V_{IL}	Input LOW Voltage	54			0.7	V	Guaranteed Input LOW Voltage for All Inputs
		74			0.8		
V_{IK}	Input Clamp Diode Voltage			−0.65	−1.5	V	V_{CC} = MIN, I_{IN} = −18 mA
V_{OH}	Ouput HIGH Voltage	54	2.5	3.5		V	V_{CC} = MIN, I_{OH} = MAX, V_{IN} = V_{IH} or V_{IL} per Truth Table
		74	2.7	3.5		V	
V_{OL}	Ouput LOW Voltage	54, 74		0.25	0.4	V	I_{OL} = 4.0 mA; V_{CC} = V_{CC} MIN, V_{IN} = V_{IL} or V_{IH} per Truth Table
		74		0.35	0.5	V	I_{OL} = 8.0 mA
I_{IH}	Input HIGH Current				20	μA	V_{CC} = MAX, V_{IN} = 2.7 V
					0.1	mA	V_{CC} = MAX, V_{IN} = 7.0 V
I_{IL}	Input LOW Current				−0.4	mA	V_{CC} = MAX, I_N = 0.4 V
I_{OS}	Short Circuit Current (Note 1)		−20		−100	mA	V_{CC} = MAX
I_{CC}	Power Supply Current Total, Output HIGH				1.6	mA	V_{CC} = MAX
	Total, Output LOW				4.4		

NOTE 1: Not more than one output should be shorted at a time, nor for more than 1 second.

AC CHARACTERISTICS (T_A = 25°C)

Symbol	Parameter	Limits Min	Typ	Max	Unit	Test Conditions
t_{PLH}	Turn-Off Delay, Input to Output		9.0	15	ns	V_{CC} = 5.0 V
t_{PHL}	Turn-On Delay, Input to Output		10	15	ns	C_L = 15 pF

GUARANTEED OPERATING RANGES

Symbol	Parameter		Min	Typ	Max	Unit
V_{CC}	Supply Voltage	54	4.5	5.0	5.5	V
		74	4.75	5.0	5.25	
T_A	Operating Ambient Temperature Range	54	−55	25	125	°C
		74	0	25	70	
I_{OH}	Output Current — High	54, 74			−0.4	mA
I_{OL}	Output Current — Low	54			4.0	mA
		74			8.0	

▲ **FIGURE 3-56**

Bipolar logic. Partial data sheet for a 54/74LS00 quad 2-input NAND gate.

Performance Characteristics and Parameters

Several things define the performance of a logic circuit. These performance characteristics are the switching speed measured in terms of the propagation delay time, the power dissipation, the fan-out or drive capability, the speed-power product, the dc supply voltage, and the input/output logic levels.

High-speed logic has a short propagation delay time.

Propagation Delay Time

This parameter is a result of the limitation on switching speed or frequency at which a logic circuit can operate. The terms *low speed* and *high speed,* applied to logic circuits, refer to the propagation delay time. The shorter the propagation delay, the higher the switching speed of the circuit and thus the higher the frequency at which it can operate.

Propagation delay time, t_P, of a logic gate is the time interval between the transition of an input pulse and the occurrence of the resulting transition of the output pulse. There are two different measurements of propagation delay time associated with a logic gate that apply to all the types of basic gates:

- t_{PHL}: The time between a specified reference point on the input pulse and a corresponding reference point on the resulting output pulse, with the output changing from the HIGH level to the LOW level (HL).
- t_{PLH}: The time between a specified reference point on the input pulse and a corresponding reference point on the resulting output pulse, with the output changing from the LOW level to the HIGH level (LH).

For the HCT family CMOS, the propagation delay is 7 ns, for the AC family it is 5 ns, and for the ALVC family it is 3 ns. For standard-family bipolar (TTL) gates, the typical propagation delay is 11 ns and for F family gates it is 3.3 ns. All specified values are dependent on certain operating conditions as stated on a data sheet.

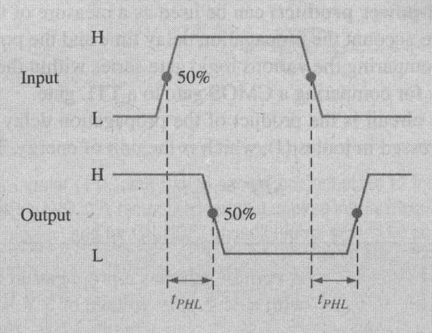
EXAMPLE 3-22

Show the propagation delay times of an inverter.

Solution

An input/output pulse of an inverter is shown in Figure 3–57, and the propagation delay times, t_{PHL} and t_{PLH}, are indicated. In this case, the delays are measured between the 50% points of the corresponding edges of the input and output pulses. The values of t_{PHL} and t_{PLH} are not necessarily equal but in many cases they are the same.

FIGURE 3-57



Related Problem

One type of logic gate has a specified maximum t_{PLH} and t_{PHL} of 10 ns. For another type of gate the value is 4 ns. Which gate can operate at the highest frequency?

DC Supply Voltage (V_{CC})

The typical dc supply voltage for CMOS logic is either 5 V, 3.3 V, 2.5 V, or 1.8 V, depending on the category. An advantage of CMOS is that the supply voltages can vary over a wider range than for bipolar logic. The 5 V CMOS can tolerate supply variations from 2 V to 6 V and still operate properly although propagation delay time and power dissipation are significantly affected. The 3.3 V CMOS can operate with supply voltages from 2 V to 3.6 V. The typical dc supply voltage for bipolar logic is 5.0 V with a minimum of 4.5 V and a maximum of 5.5 V.

Power Dissipation

A lower power dissipation means less current from the dc supply.

The **power dissipation,** P_D, of a logic gate is the product of the dc supply voltage and the average supply current. Normally, the supply current when the gate output is LOW is greater than when the gate output is HIGH. The manufacturer's data sheet usually designates the supply current for the LOW output state as I_{CCL} and for the HIGH state as I_{CCH}. The average supply current is determined based on a 50% duty cycle (output LOW half the time and HIGH half the time), so the average power dissipation of a logic gate is

$$P_D = V_{CC}\left(\frac{I_{CCH} + I_{CCL}}{2}\right) \qquad \text{Equation 3–2}$$

CMOS gates have very low power dissipations compared to the bipolar family. However, the power dissipation of CMOS is dependent on the frequency of operation. At zero frequency the quiescent power is typically in the microwatt/gate range, and at the maximum operating frequency it can be in the low milliwatt range; therefore, power is sometimes specified at a given frequency. The HC family, for example, has a power of 2.75 μW/gate at 0 Hz (quiescent) and 600 μW/gate at 1 MHz.

Power dissipation for bipolar gates is independent of frequency. For example, the ALS family uses 1.4 mW/gate regardless of the frequency and the F family uses 6 mW/gate.

Input and Output Logic Levels

V_{IL} is the LOW level input voltage for a logic gate, and V_{IH} is the HIGH level input voltage. The 5 V CMOS accepts a maximum voltage of 1.5 V as V_{IL} and a minimum voltage

of 3.5 V as V_{IH}. Bipolar logic accepts a maximum voltage of 0.8 V as V_{IL} and a minimum voltage of 2 V as V_{IH}.

V_{OL} is the LOW level output voltage and V_{OH} is the HIGH level output voltage. For 5 V CMOS, the maximum V_{OL} is 0.33 V and the minimum V_{OH} is 4.4 V. For bipolar logic, the maximum V_{OL} is 0.4 V and the minimum V_{OH} is 2.4 V. All values depend on operating conditions as specified on the data sheet.

Speed-Power Product (SPP)

This parameter (**speed-power product**) can be used as a measure of the performance of a logic circuit taking into account the propagation delay time and the power dissipation. It is especially useful for comparing the various logic gate series within the CMOS and bipolar technology families or for comparing a CMOS gate to a TTL gate.

The SPP of a logic circuit is the product of the propagation delay time and the power dissipation and is expressed in joules (J), which is the unit of energy. The formula is

$$SPP = t_p P_D \qquad \text{Equation 3–3}$$

EXAMPLE 3-23

A certain gate has a propagation delay of 5 ns and $I_{CCH} = 1$ mA and $I_{CCL} = 2.5$ mA with a dc supply voltage of 5 V. Determine the speed-power product.

Solution

$$P_D = V_{CC}\left(\frac{I_{CCH} + I_{CCL}}{2}\right) = 5\text{ V}\left(\frac{1\text{ mA} + 2.5\text{ mA}}{2}\right) = 5\text{ V}(1.75\text{ mA}) = 8.75\text{ mW}$$

$$SPP = (5\text{ ns})(8.75\text{ mW}) = \mathbf{43.75\text{ pJ}}$$

Related Problem

If the propagation delay of a gate is 15 ns and its *SPP* is 150 pJ, what is its average power dissipation?

Fan-Out and Loading

The **fan-out** of a logic gate is the maximum number of inputs of the same series in an IC family that can be connected to a gate's output and still maintain the output voltage levels within specified limits. Fan-out is a significant parameter only for bipolar logic because of the type of circuit technology. Since very high impedances are associated with CMOS circuits, the fan-out is very high but depends on frequency because of capacitive effects.

Fan-out is specified in terms of **unit loads**. A unit load for a logic gate equals one input to a like circuit. For example, a unit load for a 74LS00 NAND gate equals *one* input to another logic gate in the 74LS family (not necessarily a NAND gate). Because the current from a LOW input (I_{IL}) of a 74LS00 gate is 0.4 mA and the current that a LOW output (I_{OL}) can accept is 8.0 mA, the number of unit loads that a 74LS00 gate can drive in the LOW state is

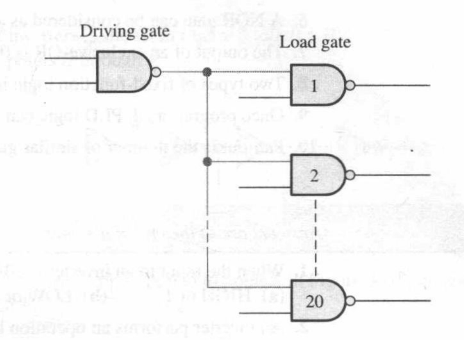
A higher fan-out means that a gate output can be connected to more gate inputs.

$$\text{Unit loads} = \frac{I_{OL}}{I_{IL}} = \frac{8.0\text{ mA}}{0.4\text{ mA}} = 20$$

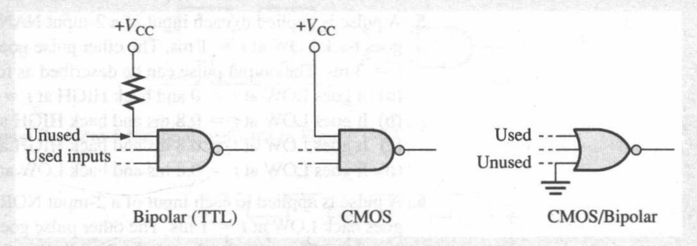
Figure 3–58 shows LS logic gates driving a number of other gates of the same circuit technology, where the number of gates depends on the particular circuit technology. For example, as you have seen, the maximum number of gate inputs (unit loads) that a 74LS family bipolar gate can drive is 20.

▶ **FIGURE 3-58**

The LS family NAND gate output fans out to a maximum of 20 LS family gate inputs.



Unused gate inputs for bipolar (TTL) and CMOS should be connected to the appropriate logic level (HIGH or LOW). For AND/NAND, it is recommended that unused inputs be connected to V_{CC} (through a 1.0 kΩ resistor with bipolar) and for OR/NOR, unused inputs should be connected to ground.



SECTION 3-7 CHECKUP

1. How is fixed-function logic different than PLD logic?
2. List the two types of IC technologies that are the most widely used.
3. Identify the following IC logic designators:
   (a) LS (b) HC (c) HCT
4. Which IC technology generally has the lowest power dissipation?
5. What does the term *hex inverter* mean? What does *quad 2-input NAND* mean?
6. A positive pulse is applied to an inverter input. The time from the leading edge of the input to the leading edge of the output is 10 ns. The time from the trailing edge of the input to the trailing edge of the output is 8 ns. What are the values of t_{PLH} and t_{PHL}?
7. A certain gate has a propagation delay time of 6 ns and a power dissipation of 3 mW. Determine the speed-power product?
8. Define I_{CCL} and I_{CCH}.
9. Define V_{IL} and V_{IH}.
10. Define V_{OL} and V_{OH}.

TRUE/FALSE QUIZ

Answers are at the end of the chapter.

1. An inverter performs the NOR operation.
2. An AND gate can have only two inputs.
3. If any input to an OR is 1, the output is 1.
4. If all inputs to an AND gate are 1, the output is 0.
5. A NAND gate has an output that is opposite the output of an AND gate.

6. A NOR gate can be considered as an OR gate followed by an inverter.
7. The output of an exclusive-OR is 0 if the inputs are opposite.
8. Two types of fixed-function logic integrated circuits are bipolar and NMOS.
9. Once programmed, PLD logic can be changed.
10. Fan-out is the number of similar gates that a given gate can drive.

SELF-TEST

Answers are at the end of the chapter.

1. When the input to an inverter is HIGH (1), the output is
   (a) HIGH or 1 **(b)** LOW or 1 **(c)** HIGH or 0 **(d)** LOW or 0
2. An inverter performs an operation known as
   (a) complementation **(b)** assertion **(c)** inversion **(d)** both answers (a) and (c)
3. The output of an AND gate with inputs A, B, and C is a 1 (HIGH) when
   (a) $A = 1, B = 1, C = 1$ **(b)** $A = 1, B = 0, C = 1$ **(c)** $A = 0, B = 0, C = 0$
4. The output of an OR gate with inputs A, B, and C is a 1 (HIGH) when
   (a) $A = 1, B = 1, C = 1$ **(b)** $A = 0, B = 0, C = 1$ **(c)** $A = 0, B = 0, C = 0$
   (d) answers (a), (b), and (c) **(e)** only answers (a) and (b)
5. A pulse is applied to each input of a 2-input NAND gate. One pulse goes HIGH at $t = 0$ and goes back LOW at $t = 1$ ms. The other pulse goes HIGH at $t = 0.8$ ms and goes back LOW at $t = 3$ ms. The output pulse can be described as follows:
   (a) It goes LOW at $t = 0$ and back HIGH at $t = 3$ ms.
   (b) It goes LOW at $t = 0.8$ ms and back HIGH at $t = 3$ ms.
   (c) It goes LOW at $t = 0.8$ ms and back HIGH at $t = 1$ ms.
   (d) It goes LOW at $t = 0.8$ ms and back LOW at $t = 1$ ms.
6. A pulse is applied to each input of a 2-input NOR gate. One pulse goes HIGH at $t = 0$ and goes back LOW at $t = 1$ ms. The other pulse goes HIGH at $t = 0.8$ ms and goes back LOW at $t = 3$ ms. The output pulse can be described as follows:
   (a) It goes LOW at $t = 0$ and back HIGH at $t = 3$ ms.
   (b) It goes LOW at $t = 0.8$ ms and back HIGH at $t = 3$ ms.
   (c) It goes LOW at $t = 0.8$ ms and back HIGH at $t = 1$ ms.
   (d) It goes HIGH at $t = 0.8$ ms and back LOW at $t = 1$ ms.
7. A pulse is applied to each input of an exclusive-OR gate. One pulse goes HIGH at $t = 0$ and goes back LOW at $t = 1$ ms. The other pulse goes HIGH at $t = 0.8$ ms and goes back LOW at $t = 3$ ms. The output pulse can be described as follows:
   (a) It goes HIGH at $t = 0$ and back LOW at $t = 3$ ms.
   (b) It goes HIGH at $t = 0$ and back LOW at $t = 0.8$ ms.
   (c) It goes HIGH at $t = 1$ ms and back LOW at $t = 3$ ms.
   (d) both answers (b) and (c)
8. A positive-going pulse is applied to an inverter. The time interval from the leading edge of the input to the leading edge of the output is 7 ns. This parameter is
   (a) speed-power product **(b)** propagation delay, t_{PHL}
   (c) propagation delay, t_{PLH} **(d)** pulse width
9. The purpose of a programmable link in an AND array is to
   (a) connect an input variable to a gate input
   (b) connect a row to a column in the array matrix
   (c) disconnect a row from a column in the array matrix
   (d) do all of the above

PROBLEMS

Answers to odd-numbered problems are at the end of the book.

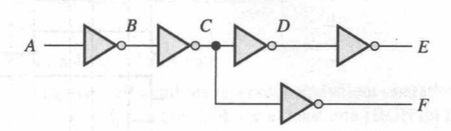
Section 3–1 The Inverter

1. The input waveform shown in Figure 3–59 is applied to an inverter. Draw the timing diagram of the output waveform in proper relation to the input.

▶ **FIGURE 3-59**

2. A combination of inverters is shown in Figure 3–60. If a HIGH is applied to point *A*, determine the logic levels at points *B* through *F*.

▶ FIGURE 3-60

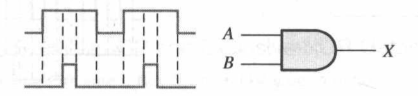


3. If the waveform in Figure 3–59 is applied to point *A* in Figure 3–77, determine the waveforms at points *B* through *F*.

Section 3–2 The AND Gate

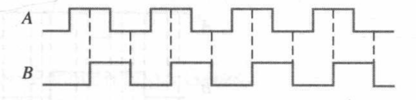
4. Draw the rectangular outline symbol for a 4-input AND gate.
5. Determine the output, *X*, for a 2-input AND gate with the input waveforms shown in Figure 3–61. Show the proper relationship of output to inputs with a timing diagram.

▶ FIGURE 3-61



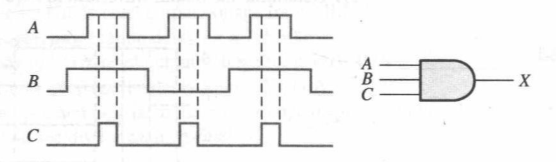
6. Repeat Problem 5 for the waveforms in Figure 3–62.

▶ FIGURE 3-62



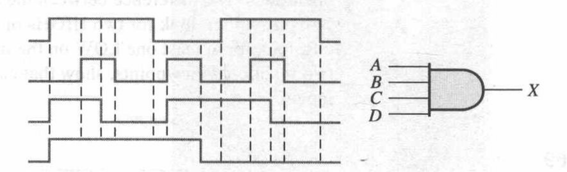
7. The input waveforms applied to a 3-input AND gate are as indicated in Figure 3–63. Show the output waveform in proper relation to the inputs with a timing diagram.

▶ FIGURE 3-63



8. The input waveforms applied to a 4-input AND gate are as indicated in Figure 3–64. Show the output waveform in proper relation to the inputs with a timing diagram.

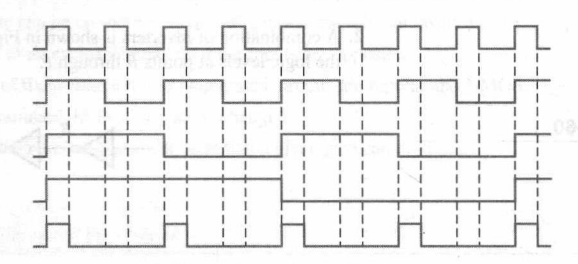
▶ FIGURE 3-64



Section 3–3 The OR Gate

9. Draw the rectangular outline symbol for a 3-input AND gate.
10. Write the expression for a 5-input OR gate with inputs *A*, *B*, *C*, *D*, *E*, and output *X*.
11. Determine the output for a 2-input OR gate when the input waveforms are as in Figure 3–62 and draw a timing diagram.
12. Repeat Problem 7 for a 3-input OR gate.
13. Repeat Problem 8 for a 4-input OR gate.
14. For the five input waveforms in Figure 3–65, determine the output if the five signals are ANDed. Determine the output if the five signals are ORed. Draw the timing diagram for each case.

▶ FIGURE 3-65



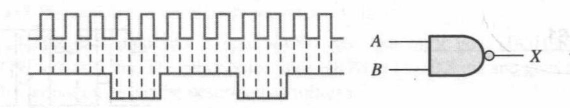
15. Draw the rectangular outline symbol for a 4-input OR gate.

16. Show the truth table for a 3-input OR gate.

Section 3–4 The NAND Gate

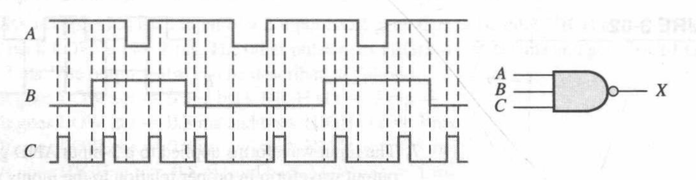
17. For the set of input waveforms in Figure 3–66, determine the output for the gate shown and draw the timing diagram.

▶ FIGURE 3-66



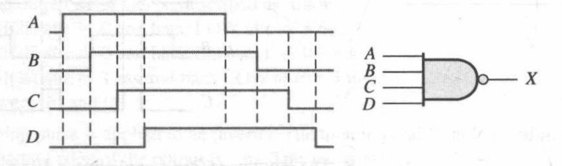
18. Determine the gate output for the input waveforms in Figure 3–67 and draw the timing diagram.

▶ FIGURE 3-67



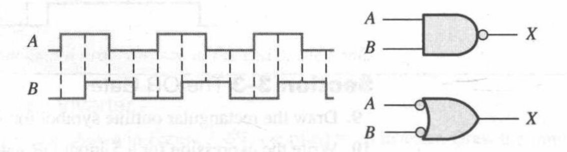
19. Determine the output waveform in Figure 3–68.

▶ FIGURE 3-68



20. As you have learned, the two logic symbols shown in Figure 3–69 represent equivalent operations. The difference between the two is strictly from a functional viewpoint. For the NAND symbol, look for two HIGHs on the inputs to give a LOW output. For the negative-OR, look for at least one LOW on the inputs to give a HIGH on the output. Using these two functional viewpoints, show that each gate will produce the same output for the given inputs.

▶ FIGURE 3-69

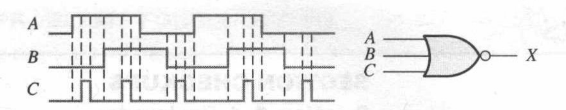


Section 3–5 The NOR Gate

21. Repeat Problem 17 for a 2-input NOR gate.

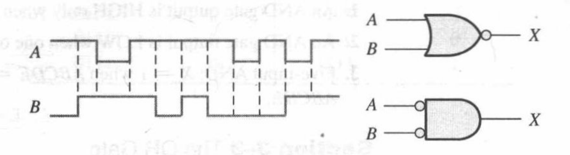
22. Determine the output waveform in Figure 3–70 and draw the timing diagram.

▶ FIGURE 3-70



23. Repeat Problem 19 for a 4-input NOR gate.
24. The NAND and the negative-OR symbols represent equivalent operations, but they are functionally different. For the NOR symbol, look for at least one HIGH on the inputs to give a LOW on the output. For the negative-AND, look for two LOWs on the inputs to give a HIGH output. Using these two functional points of view, show that both gates in Figure 3–71 will produce the same output for the given inputs.

▶ FIGURE 3-71



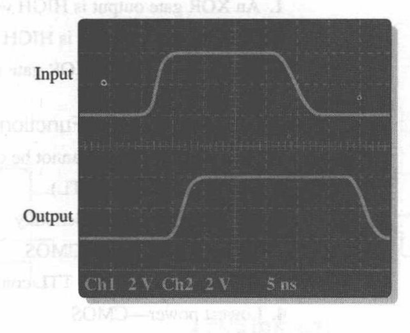
Section 3–6 The Exclusive-OR and Exclusive-NOR Gates

25. How does an exclusive-OR gate differ from an OR gate in its logical operation?
26. Repeat Problem 17 for an exclusive-OR gate.
27. Repeat Problem 17 for an exclusive-NOR gate.
28. Determine the output of an exclusive-OR gate for the inputs shown in Figure 3–62 and draw a timing diagram.

Section 3–7 Fixed-Function Logic Gates

29. In the comparison of certain logic devices, it is noted that the power dissipation for one particular type increases as the frequency increases. Is the device bipolar or CMOS?
30. Using the data sheets in Figures 3–55 and 3–56, determine the following:
    (a) 74LS00 power dissipation at maximum supply voltage and a 50% duty cycle
    (b) Minimum HIGH level output voltage for a 74LS00
    (c) Maximum propagation delay for a 74LS00
    (d) Maximum LOW level output voltage for a 74HC00A
    (e) Maximum propagation delay for a 74HC00A
31. Determine t_{PLH} and t_{PHL} from the oscilloscope display in Figure 3–72. The readings indicate volts/div and sec/div for each channel.

▶ FIGURE 3-72



32. Gate *A* has $t_{PLH} = t_{PHL} = 6$ ns. Gate *B* has $t_{PLH} = t_{PHL} = 10$ ns. Which gate can be operated at a higher frequency?
33. If a logic gate operates on a dc supply voltage of +5 V and draws an average current of 4 mA, what is its power dissipation?
34. The variable I_{CCH} represents the dc supply current from V_{CC} when all outputs of an IC are HIGH. The variable I_{CCL} represents the dc supply current when all outputs are LOW. For a 74LS00 IC, determine the typical power dissipation when all four gate outputs are HIGH. (See data sheet in Figure 3–56.)

ANSWERS

SECTION CHECKUPS

Section 3–1 The Inverter

1. When the inverter input is 1, the output is 0.
2. (a) [inverter symbol]

   (b) A negative-going pulse is on the output (HIGH to LOW and back HIGH).

Section 3–2 The AND Gate

1. An AND gate output is HIGH only when all inputs are HIGH.
2. An AND gate output is LOW when one or more inputs are LOW.
3. Five-input AND: $X = 1$ when $ABCDE = 11111$, and $X = 0$ for all other combinations of $ABCDE$.

Section 3–3 The OR Gate

1. An OR gate output is HIGH when one or more inputs are HIGH.
2. An OR gate output is LOW only when all inputs are LOW.
3. Three-input OR: $X = 0$ when $ABC = 000$, and $X = 1$ for all other combinations of ABC.

Section 3–4 The NAND Gate

1. A NAND gate output is LOW only when all inputs are HIGH.
2. A NAND gate output is HIGH when one or more inputs are LOW.
3. NAND: active-LOW output for all HIGH inputs; negative-OR: active-HIGH output for one or more LOW inputs. They have the same truth tables.
4. $X = \overline{ABC}$

Section 3–5 The NOR Gate

1. A NOR gate output is HIGH only when all inputs are LOW.
2. A NOR gate output is LOW when one or more inputs are HIGH.
3. NOR: active-LOW output for one or more HIGH inputs; negative-AND: active-HIGH output for all LOW inputs. They have the same truth tables.
4. $X = \overline{A + B + C}$

Section 3–6 The Exclusive-OR and Exclusive-NOR Gates

1. An XOR gate output is HIGH when the inputs are at opposite levels.
2. An XNOR gate output is HIGH when the inputs are at the same levels.
3. Apply the bits to the XOR gate inputs; when the output is HIGH, the bits are different.

Section 3–7 Fixed-Function Logic Gates

1. Fixed-function logic cannot be changed. PLDs can be programmed for any logic function.
2. CMOS and bipolar (TTL)
3. (a) LS—Low-power Schottky

   (b) HC—High-speed CMOS

   (c) HCT—HC CMOS TTL compatible
4. Lowest power—CMOS
5. Six inverters in a package; four 2-input NAND gates in a package
6. $t_{PLH} = 10$ ns; $t_{PHL} = 8$ ns
7. 18 pJ
8. I_{CCL}—dc supply current for LOW output state; I_{CCH}—dc supply current for HIGH output state
9. V_{IL}—LOW input voltage; V_{IH}—HIGH input voltage
10. V_{OL}—LOW output voltage; V_{OH}—HIGH output voltage

RELATED PROBLEMS FOR EXAMPLES

3–1 The timing diagram is not affected.

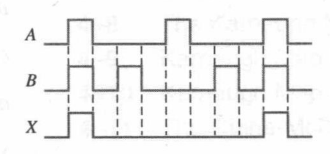
3–2 See Table 3–15.

▶ **TABLE 3-15**

Inputs *ABCD*	Output *X*	Inputs *ABCD*	Output *X*
0000	0	1000	0
0001	0	1001	0
0010	0	1010	0
0011	0	1011	0
0100	0	1100	0
0101	0	1101	0
0110	0	1110	0
0111	0	1111	1

3–3 See Figure 3–73.

▶ **FIGURE 3-73**

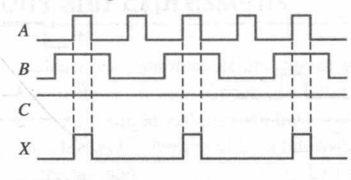


3–4 The output waveform is the same as input *A*.

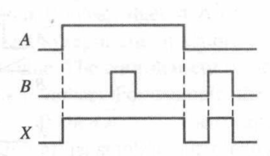
3–5 See Figure 3–74.

3–6 Results are the same as example.

3–7 See Figure 3–75.



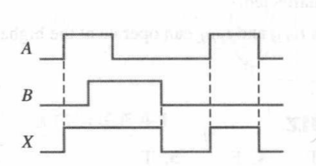
▲ **FIGURE 3-74**



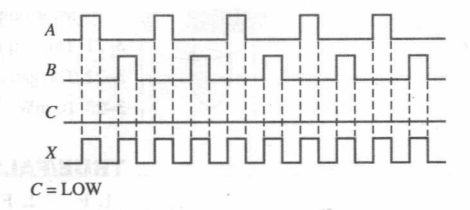
▲ **FIGURE 3-75**

3–8 See Figure 3–76.

3–9 See Figure 3–77.



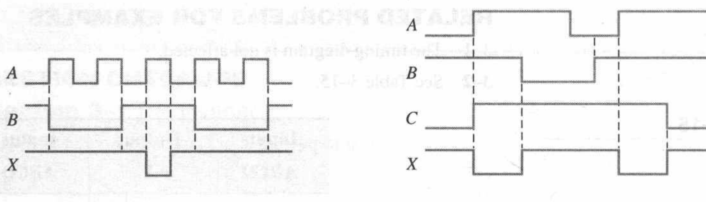
▲ **FIGURE 3-76**



▲ **FIGURE 3-77**

3–10 See Figure 3–78.

3–11 See Figure 3–79.



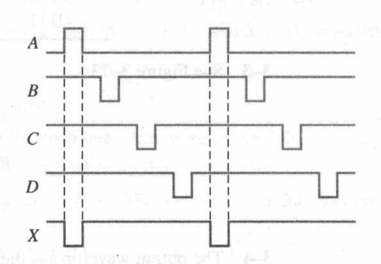
▲FIGURE 3-78 ▲FIGURE 3-79

3–12 Use a 3-input NAND gate.

3–13 Use a 4-input NAND gate operating as a negative-OR gate.

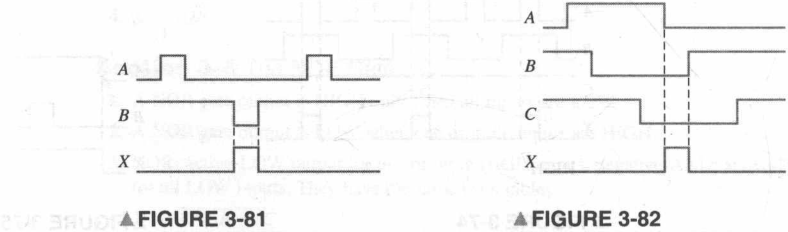
3–14 See Figure 3–80.

▶FIGURE 3-80



3–15 See Figure 3–81.

3–16 See Figure 3–82.



▲FIGURE 3-81 ▲FIGURE 3-82

3–17 Use a 2-input NOR gate.

3–18 A 3-input NAND gate.

3–19 The output is always LOW. The output is a straight line.

3–20 The exclusive-OR gate will not detect simultaneous failures if both circuits produce the same outputs.

3–21 The outputs are unaffected.

3–22 The gate with 4 ns t_{PLH} and t_{PHL} can operate at the highest frequency.

3–23 10 mW

TRUE/FALSE QUIZ

1. F **2.** F **3.** T **4.** F **5.** T
6. T **7.** F **8.** F **9.** T **10.** T

SELF-TEST

1. (d) **2.** (d) **3.** (a) **4.** (e) **5.** (c) **6.** (a) **7.** (d) **8.** (b) **9.** (d)

Chapter 4 Boolean Algebra and Logic Simplification

CHAPTER OUTLINE

4-1 Boolean Operations and Expressions

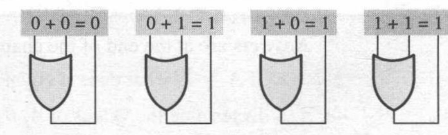
InfoNote

In a microprocessor, the arithmetic logic unit (ALU) performs arithmetic and Boolean logic operations on digital data as directed by program instructions. Logical operations are equivalent to the basic gate operations that you are familiar with but deal with a minimum of 8 bits at a time. Examples of Boolean logic instructions are AND, OR, NOT, and XOR, which are called *mnemonics*. An assembly language program uses the mnemonics to specify an operation. Another program called an *assembler* translates the mnemonics into a binary code that can be understood by the microprocessor.

Variable, complement, and *literal* are terms used in Boolean algebra. A **variable** is a symbol (usually an italic uppercase letter or word) used to represent an action, a condition, or data. Any single variable can have only a 1 or a 0 value. The **complement** is the inverse of a variable and is indicated by a bar over the variable (overbar). For example, the complement of the variable A is $\overline{A}$. If $A = 1$, then $\overline{A} = 0$. If $A = 0$, then $\overline{A} = 1$. The complement of the variable A is read as "not A" or "A bar." Sometimes a prime symbol rather than an overbar is used to denote the complement of a variable; for example, B' indicates the complement of B. In this book, only the overbar is used. A **literal** is a variable or the complement of a variable.

Boolean Addition

Recall from Chapter 3 that **Boolean addition** is equivalent to the OR operation. The basic rules are illustrated with their relation to the OR gate in Figure 4–1.



▲FIGURE 4-1

The OR operation is the Boolean equivalent of addition.

In Boolean algebra, a **sum term** is a sum of literals. In logic circuits, a sum term is produced by an OR operation with no AND operations involved. Some examples of sum terms are $A + B$, $A + \overline{B}$, $A + B + \overline{C}$, and $\overline{A} + B + C + \overline{D}$.

A sum term is equal to 1 when one or more of the literals in the term are 1. A sum term is equal to 0 only if each of the literals is 0.

EXAMPLE 4-1

Determine the values of A, B, C, and D that make the sum term $A + \overline{B} + C + \overline{D}$ equal to 0.

Solution

For the sum term to be 0, each of the literals in the term must be 0. Therefore, $A = \mathbf{0}$, $B = \mathbf{1}$ so that $\overline{B} = 0$, $C = \mathbf{0}$, and $D = \mathbf{1}$ so that $\overline{D} = 0$.

$$A + \overline{B} + C + \overline{D} = 0 + \overline{1} + 0 + \overline{1} = 0 + 0 + 0 + 0 = 0$$

Related Problem*

Determine the values of A and B that make the sum term $\overline{A} + B$ equal to 0.

*Answers are at the end of the chapter.

Boolean Multiplication

Also recall from Chapter 3 that **Boolean multiplication** is equivalent to the AND operation. The basic rules are illustrated with their relation to the AND gate in Figure 4–2.

The AND operation is the Boolean equivalent of multiplication.

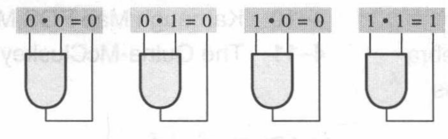


FIGURE 4-2

In Boolean algebra, a **product term** is the product of literals. In logic circuits, a product term is produced by an AND operation with no OR operations involved. Some examples of product terms are AB, $A\overline{B}$, ABC, and $A\overline{B}C\overline{D}$.

A product term is equal to 1 only if each of the literals in the term is 1. A product term is equal to 0 when one or more of the literals are 0.

EXAMPLE 4-2

Determine the values of A, B, C, and D that make the product term $A\overline{B}C\overline{D}$ equal to 1.

Solution

For the product term to be 1, each of the literals in the term must be 1. Therefore, $A = \mathbf{1}$, $B = \mathbf{0}$ so that $\overline{B} = 1$, $C = \mathbf{1}$, and $D = \mathbf{0}$ so that $\overline{D} = 1$.

$$A\overline{B}C\overline{D} = 1 \cdot \overline{0} \cdot 1 \cdot \overline{0} = 1 \cdot 1 \cdot 1 \cdot 1 = 1$$

Related Problem

Determine the values of A and B that make the product term $\overline{A}\,\overline{B}$ equal to 1.

SECTION 4-1 CHECKUP

Answers are at the end of the chapter.

1. If $A = 0$, what does $\overline{A}$ equal?
2. Determine the values of A, B, and C that make the sum term $\overline{A} + \overline{B} + C$ equal to 0.
3. Determine the values of A, B, and C that make the product term $A\overline{B}C$ equal to 1.

4-2 Laws and Rules of Boolean Algebra

Laws of Boolean Algebra

The basic laws of Boolean algebra—the **commutative laws** for addition and multiplication, the **associative laws** for addition and multiplication, and the **distributive law**—are the same as in ordinary algebra. Each of the laws is illustrated with two or three variables, but the number of variables is not limited to this.

Commutative Laws

The *commutative law of addition* for two variables is written as

$$A + B = B + A \qquad \text{Equation 4–1}$$

This law states that the order in which the variables are ORed makes no difference. Remember, in Boolean algebra as applied to logic circuits, addition and the OR operation are the same. Figure 4–3 illustrates the commutative law as applied to the OR gate and shows that it doesn't matter to which input each variable is applied. (The symbol ≡ means "equivalent to.")

▶ FIGURE 4-3
Application of commutative law of addition.

The *commutative law of multiplication* for two variables is

$$AB = BA \qquad \text{Equation 4–2}$$

This law states that the order in which the variables are ANDed makes no difference. Figure 4–4 illustrates this law as applied to the AND gate. Remember, in Boolean algebra as applied to logic circuits, multiplication and the AND function are the same.

▶ FIGURE 4-4
Application of commutative law of multiplication.

Associative Laws

The *associative law of addition* is written as follows for three variables:

$$A + (B + C) = (A + B) + C \qquad \text{Equation 4–3}$$

This law states that when ORing more than two variables, the result is the same regardless of the grouping of the variables. Figure 4–5 illustrates this law as applied to 2-input OR gates.

▶ FIGURE 4-5
Application of associative law of addition. Open file F04-05 to verify. *A Multisim tutorial is available on the website.*

MultiSim

The *associative law of multiplication* is written as follows for three variables:

$$A(BC) = (AB)C \qquad \text{Equation 4–4}$$

This law states that it makes no difference in what order the variables are grouped when ANDing more than two variables. Figure 4–6 illustrates this law as applied to 2-input AND gates.

▶ FIGURE 4-6
Application of associative law of multiplication. Open file F04-06 to verify.

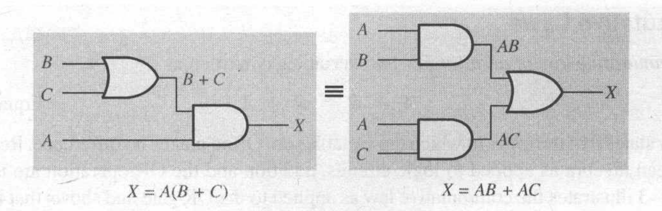
MultiSim

Distributive Law

The distributive law is written for three variables as follows:

$$A(B + C) = AB + AC \qquad \text{Equation 4–5}$$

This law states that ORing two or more variables and then ANDing the result with a single variable is equivalent to ANDing the single variable with each of the two or more variables and then ORing the products. The distributive law also expresses the process of *factoring* in which the common variable A is factored out of the product terms, for example, $AB + AC = A(B + C)$. Figure 4–7 illustrates the distributive law in terms of gate implementation.



$X = A(B + C)$ $X = AB + AC$

◄**FIGURE 4-7**

Application of distributive law. Open file F04-07 to verify.

MultiSim

Rules of Boolean Algebra

Table 4–1 lists 12 basic rules that are useful in manipulating and simplifying **Boolean expressions**. Rules 1 through 9 will be viewed in terms of their application to logic gates. Rules 10 through 12 will be derived in terms of the simpler rules and the laws previously discussed.

◄**TABLE 4-1**

Basic rules of Boolean algebra.

1. $A + 0 = A$	**7.** $A \cdot A = A$
2. $A + 1 = 1$	**8.** $A \cdot \overline{A} = 0$
3. $A \cdot 0 = 0$	**9.** $\overline{\overline{A}} = A$
4. $A \cdot 1 = A$	**10.** $A + AB = A$
5. $A + A = A$	**11.** $A + \overline{A}B = A + B$
6. $A + \overline{A} = 1$	**12.** $(A + B)(A + C) = A + BC$

A, *B*, or *C* can represent a single variable or a combination of variables.

Rule 1: $A + 0 = A$ A variable ORed with 0 is always equal to the variable. If the input variable *A* is 1, the output variable *X* is 1, which is equal to *A*. If *A* is 0, the output is 0, which is also equal to *A*. This rule is illustrated in Figure 4–8, where the lower input is fixed at 0.

$X = A + 0 = A$

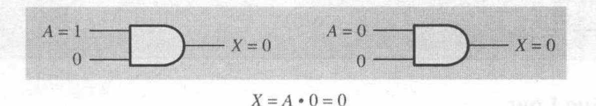
◄**FIGURE 4-8**

Rule 2: $A + 1 = 1$ A variable ORed with 1 is always equal to 1. A 1 on an input to an OR gate produces a 1 on the output, regardless of the value of the variable on the other input. This rule is illustrated in Figure 4–9, where the lower input is fixed at 1.

$X = A + 1 = 1$

◄**FIGURE 4-9**

Rule 3: $A \cdot 0 = 0$ A variable ANDed with 0 is always equal to 0. Any time one input to an AND gate is 0, the output is 0, regardless of the value of the variable on the other input. This rule is illustrated in Figure 4–10, where the lower input is fixed at 0.



$X = A \cdot 0 = 0$

◄**FIGURE 4-10**

Rule 4: $A \cdot 1 = A$ A variable ANDed with 1 is always equal to the variable. If *A* is 0, the output of the AND gate is 0. If *A* is 1, the output of the AND gate is 1 because both inputs are now 1s. This rule is shown in Figure 4–11, where the lower input is fixed at 1.

$X = A \cdot 1 = A$

◄**FIGURE 4-11**

Rule 5: $A + A = A$ A variable ORed with itself is always equal to the variable. If A is 0, then $0 + 0 = 0$; and if A is 1, then $1 + 1 = 1$. This is shown in Figure 4–12, where both inputs are the same variable.

▶ FIGURE 4-12

$X = A + A = A$

Rule 6: $A + \bar{A} = 1$ A variable ORed with its complement is always equal to 1. If A is 0, then $0 + \bar{0} = 0 + 1 = 1$. If A is 1, then $1 + \bar{1} = 1 + 0 = 1$. See Figure 4–13, where one input is the complement of the other.

▶ FIGURE 4-13

$X = A + \bar{A} = 1$

Rule 7: $A \cdot A = A$ A variable ANDed with itself is always equal to the variable. If $A = 0$, then $0 \cdot 0 = 0$; and if $A = 1$, then $1 \cdot 1 = 1$. Figure 4–14 illustrates this rule.

▶ FIGURE 4-14

$X = A \cdot A = A$

Rule 8: $A \cdot \bar{A} = 0$ A variable ANDed with its complement is always equal to 0. Either A or $\bar{A}$ will always be 0; and when a 0 is applied to the input of an AND gate, the output will be 0 also. Figure 4–15 illustrates this rule.

▶ FIGURE 4-15

$X = A \cdot \bar{A} = 0$

Rule 9: $\bar{\bar{A}} = A$ The double complement of a variable is always equal to the variable. If you start with the variable A and complement (invert) it once, you get $\bar{A}$. If you then take $\bar{A}$ and complement (invert) it, you get A, which is the original variable. This rule is shown in Figure 4–16 using inverters.

▶ FIGURE 4-16

$\bar{\bar{A}} = A$

Rule 10: $A + AB = A$ This rule can be proved by applying the distributive law, rule 2, and rule 4 as follows:

$$\begin{aligned} A + AB &= A \cdot 1 + AB = A(1 + B) && \text{Factoring (distributive law)} \\ &= A \cdot 1 && \text{Rule 2: } (1 + B) = 1 \\ &= A && \text{Rule 4: } A \cdot 1 = A \end{aligned}$$

The proof is shown in Table 4–2, which shows the truth table and the resulting logic circuit simplification.

A	B	AB	$A + AB$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

equal

A, B; A straight connection

MultiSim

◄TABLE 4-2

Rule 10: $A + AB = A$. Open file T04-02 to verify.

Rule 11: $A + \overline{A}B = A + B$ This rule can be proved as follows:

$$\begin{aligned} A + \overline{A}B &= (A + AB) + \overline{A}B && \text{Rule 10: } A = A + AB \\ &= (AA + AB) + \overline{A}B && \text{Rule 7: } A = AA \\ &= AA + AB + A\overline{A} + \overline{A}B && \text{Rule 8: adding } A\overline{A} = 0 \\ &= (A + \overline{A})(A + B) && \text{Factoring} \\ &= 1 \cdot (A + B) && \text{Rule 6: } A + \overline{A} = 1 \\ &= A + B && \text{Rule 4: drop the 1} \end{aligned}$$

The proof is shown in Table 4–3, which shows the truth table and the resulting logic circuit simplification.

A	B	$\overline{A}B$	$A + \overline{A}B$	$A + B$
0	0	0	0	0
0	1	1	1	1
1	0	0	1	1
1	1	0	1	1

equal

A, B; A, B

MultiSim

◄TABLE 4-3

Rule 11: $A + \overline{A}B = A + B$. Open file T04-03 to verify.

Rule 12: $(A + B)(A + C) = A + BC$ This rule can be proved as follows:

$$\begin{aligned} (A + B)(A + C) &= AA + AC + AB + BC && \text{Distributive law} \\ &= A + AC + AB + BC && \text{Rule 7: } AA = A \\ &= A(1 + C) + AB + BC && \text{Factoring (distributive law)} \\ &= A \cdot 1 + AB + BC && \text{Rule 2: } 1 + C = 1 \\ &= A(1 + B) + BC && \text{Factoring (distributive law)} \\ &= A \cdot 1 + BC && \text{Rule 2: } 1 + B = 1 \\ &= A + BC && \text{Rule 4: } A \cdot 1 = A \end{aligned}$$

The proof is shown in Table 4–4, which shows the truth table and the resulting logic circuit simplification.

▼TABLE 4-4

Rule 12: $(A + B)(A + C) = A + BC$. Open file T04-04 to verify.

MultiSim

A	B	C	$A + B$	$A + C$	$(A + B)(A + C)$	BC	$A + BC$
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	1	1	1	1
1	0	0	1	1	1	0	1
1	0	1	1	1	1	0	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

equal

A, B, C; A, B, C

SECTION 4-2 CHECKUP

1. Apply the associative law of addition to the expression $A + (B + C + D)$.
2. Apply the distributive law to the expression $A(B + C + D)$.

4-3 DeMorgan's Theorems

To apply DeMorgan's theorem, break the bar over the product of variables and change the sign from AND to OR.

DeMorgan's first theorem is stated as follows:

The complement of a product of variables is equal to the sum of the complements of the variables.

Stated another way,

The complement of two or more ANDed variables is equivalent to the OR of the complements of the individual variables.

The formula for expressing this theorem for two variables is

$$\overline{XY} = \overline{X} + \overline{Y} \qquad \text{Equation 4–6}$$

DeMorgan's second theorem is stated as follows:

The complement of a sum of variables is equal to the product of the complements of the variables.

Stated another way,

The complement of two or more ORed variables is equivalent to the AND of the complements of the individual variables.

The formula for expressing this theorem for two variables is

$$\overline{X + Y} = \overline{X}\,\overline{Y} \qquad \text{Equation 4–7}$$

Figure 4–17 shows the gate equivalencies and truth tables for Equations 4–6 and 4–7.

As stated, DeMorgan's theorems also apply to expressions in which there are more than two variables. The following examples illustrate the application of DeMorgan's theorems to 3-variable and 4-variable expressions.

X, Y → NAND → $\overline{XY}$ ≡ X, Y → Negative-OR → $\overline{X} + \overline{Y}$

Inputs		Output	
X	Y	$\overline{XY}$	$\overline{X} + \overline{Y}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

X, Y → NOR → $\overline{X + Y}$ ≡ X, Y → Negative-AND → $\overline{X}\,\overline{Y}$

Inputs		Output	
X	Y	$\overline{X + Y}$	$\overline{X}\,\overline{Y}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

▶ **FIGURE 4-17** Gate equivalencies and the corresponding truth tables that illustrate DeMorgan's theorems. Notice the equality of the two output columns in each table. This shows that the equivalent gates perform the same logic function.

EXAMPLE 4-3

Apply DeMorgan's theorems to the expressions $\overline{XYZ}$ and $\overline{X + Y + Z}$.

Solution

$$\overline{XYZ} = \overline{X} + \overline{Y} + \overline{Z}$$
$$\overline{X + Y + Z} = \overline{X}\,\overline{Y}\,\overline{Z}$$

Related Problem

Apply DeMorgan's theorem to the expression $\overline{\overline{X} + \overline{Y} + \overline{Z}}$.

EXAMPLE 4-4

Apply DeMorgan's theorems to the expressions $\overline{WXYZ}$ and $\overline{W + X + Y + Z}$.

Solution

$$\overline{WXYZ} = \overline{W} + \overline{X} + \overline{Y} + \overline{Z}$$
$$\overline{W + X + Y + Z} = \overline{W}\,\overline{X}\,\overline{Y}\,\overline{Z}$$

Related Problem

Apply DeMorgan's theorem to the expression $\overline{\overline{W}\,\overline{X}\,\overline{Y}\,\overline{Z}}$.

Each variable in DeMorgan's theorems as stated in Equations 4–6 and 4–7 can also represent a combination of other variables. For example, X can be equal to the term $AB + C$, and Y can be equal to the term $A + BC$. So if you can apply DeMorgan's theorem for two variables as stated by $\overline{XY} = \overline{X} + \overline{Y}$ to the expression $\overline{(AB + C)(A + BC)}$, you get the following result:

$$\overline{(AB + C)(A + BC)} = (\overline{AB + C}) + (\overline{A + BC})$$

Notice that in the preceding result you have two terms, $\overline{AB + C}$ and $\overline{A + BC}$, to each of which you can again apply DeMorgan's theorem $\overline{X + Y} = \overline{X}\,\overline{Y}$ individually, as follows:

$$(\overline{AB + C}) + (\overline{A + BC}) = (\overline{AB})\overline{C} + \overline{A}(\overline{BC})$$

Notice that you still have two terms in the expression to which DeMorgan's theorem can again be applied. These terms are $\overline{AB}$ and $\overline{BC}$. A final application of DeMorgan's theorem gives the following result:

$$(\overline{AB})\overline{C} + \overline{A}(\overline{BC}) = (\overline{A} + \overline{B})\overline{C} + \overline{A}(\overline{B} + \overline{C})$$

Although this result can be simplified further by the use of Boolean rules and laws, DeMorgan's theorems cannot be used any more.

Applying DeMorgan's Theorems

The following procedure illustrates the application of DeMorgan's theorems and Boolean algebra to the specific expression

$$\overline{\overline{A + B\overline{C}} + D(\overline{E + \overline{F}})}$$

Step 1: Identify the terms to which you can apply DeMorgan's theorems, and think of each term as a single variable. Let $\overline{A + B\overline{C}} = X$ and $D(\overline{E + \overline{F}}) = Y$.

Step 2: Since $\overline{X + Y} = \overline{X}\,\overline{Y}$,

$$\overline{(\overline{A + B\overline{C}}) + (D(\overline{E + \overline{F}}))} = (\overline{\overline{A + B\overline{C}}})(\overline{D(\overline{E + \overline{F}})})$$

Step 3: Use rule 9 ($\overline{\overline{A}} = A$) to cancel the double bars over the left term (this is not part of DeMorgan's theorem).

$$(\overline{\overline{A + B\overline{C}}})(\overline{D(\overline{E + \overline{F}})}) = (A + B\overline{C})(\overline{D(\overline{E + \overline{F}})})$$

Step 4: Apply DeMorgan's theorem to the second term.

$$(A + B\overline{C})(\overline{D(\overline{E + \overline{F}})}) = (A + B\overline{C})(\overline{D} + (\overline{\overline{E + \overline{F}}}))$$

Step 5: Use rule 9 ($\overline{A} = A$) to cancel the double bars over the $E + \overline{F}$ part of the term.

$$(A + B\overline{C})(\overline{D} + \overline{\overline{E + \overline{F}}}) = (A + B\overline{C})(\overline{D} + E + \overline{F})$$

The following three examples will further illustrate how to use DeMorgan's theorems.

EXAMPLE 4-5

Apply DeMorgan's theorems to each of the following expressions:

(a) $\overline{(A + B + C)D}$

(b) $\overline{ABC + DEF}$

(c) $\overline{A\overline{B} + \overline{C}D + EF}$

Solution

(a) Let $A + B + C = X$ and $D = Y$. The expression $\overline{(A + B + C)D}$ is of the form $\overline{XY} = \overline{X} + \overline{Y}$ and can be rewritten as

$$\overline{(A + B + C)D} = \overline{A + B + C} + \overline{D}$$

Next, apply DeMorgan's theorem to the term $\overline{A + B + C}$.

$$\overline{A + B + C} + \overline{D} = \overline{A}\,\overline{B}\,\overline{C} + \overline{D}$$

(b) Let $ABC = X$ and $DEF = Y$. The expression $\overline{ABC + DEF}$ is of the form $\overline{X + Y} = \overline{X}\,\overline{Y}$ and can be rewritten as

$$\overline{ABC + DEF} = (\overline{ABC})(\overline{DEF})$$

Next, apply DeMorgan's theorem to each of the terms $\overline{ABC}$ and $\overline{DEF}$.

$$(\overline{ABC})(\overline{DEF}) = (\overline{A} + \overline{B} + \overline{C})(\overline{D} + \overline{E} + \overline{F})$$

(c) Let $A\overline{B} = X$, $\overline{C}D = Y$, and $EF = Z$. The expression $\overline{A\overline{B} + \overline{C}D + EF}$ is of the form $\overline{X + Y + Z} = \overline{X}\,\overline{Y}\,\overline{Z}$ and can be rewritten as

$$\overline{A\overline{B} + \overline{C}D + EF} = (\overline{A\overline{B}})(\overline{\overline{C}D})(\overline{EF})$$

Next, apply DeMorgan's theorem to each of the terms $\overline{A\overline{B}}$, $\overline{\overline{C}D}$, and $\overline{EF}$.

$$(\overline{A\overline{B}})(\overline{\overline{C}D})(\overline{EF}) = (\overline{A} + B)(C + \overline{D})(\overline{E} + \overline{F})$$

Related Problem

Apply DeMorgan's theorems to the expression $\overline{\overline{ABC} + D + E}$.

EXAMPLE 4-6

Apply DeMorgan's theorems to each expression:

(a) $\overline{(\overline{A + B}) + \overline{C}}$

(b) $\overline{(\overline{A} + B) + CD}$

(c) $\overline{(A + B)\overline{C}\,\overline{D} + E + \overline{F}}$

Solution

(a) $\overline{(\overline{A + B}) + \overline{C}} = (\overline{\overline{A + B}})\overline{\overline{C}} = (A + B)C$

(b) $\overline{(\overline{A} + B) + CD} = (\overline{\overline{A} + B})\overline{CD} = (\overline{\overline{A}}\,\overline{B})(\overline{C} + \overline{D}) = A\overline{B}(\overline{C} + \overline{D})$

(c) $\overline{(A + B)\overline{C}\,\overline{D} + E + \overline{F}} = (\overline{(A + B)\overline{C}\,\overline{D}})(\overline{E + \overline{F}}) = (\overline{A}\,\overline{B} + C + D)\overline{E}F$

Related Problem

Apply DeMorgan's theorems to the expression $\overline{\overline{A}B(C + \overline{D}) + E}$.

EXAMPLE 4-7

The Boolean expression for an exclusive-OR gate is $A\overline{B} + \overline{A}B$. With this as a starting point, use DeMorgan's theorems and any other rules or laws that are applicable to develop an expression for the exclusive-NOR gate.

Solution

Start by complementing the exclusive-OR expression and then applying DeMorgan's theorems as follows:

$$\overline{A\overline{B} + \overline{A}B} = (\overline{A\overline{B}})(\overline{\overline{A}B}) = (\overline{A} + \overline{\overline{B}})(\overline{\overline{A}} + \overline{B}) = (\overline{A} + B)(A + \overline{B})$$

Next, apply the distributive law and rule 8 ($A \cdot \overline{A} = 0$).

$$(\overline{A} + B)(A + \overline{B}) = \overline{A}A + \overline{A}\,\overline{B} + AB + B\overline{B} = \overline{A}\,\overline{B} + AB$$

The final expression for the XNOR is $\overline{A}\,\overline{B} + AB$. Note that this expression equals 1 any time both variables are 0s or both variables are 1s.

Related Problem

Starting with the expression for a 4-input NAND gate, use DeMorgan's theorems to develop an expression for a 4-input negative-OR gate.

SECTION 4-3 CHECKUP

1. Apply DeMorgan's theorems to the following expressions:
   (a) $\overline{ABC} + (\overline{\overline{D} + E})$ (b) $\overline{(A + B)C}$ (c) $\overline{A + B + C} + \overline{\overline{D}E}$

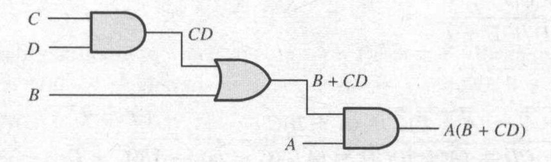
4-4 Boolean Analysis of Logic Circuits

Boolean Expression for a Logic Circuit

To derive the Boolean expression for a given combinational logic circuit, begin at the left-most inputs and work toward the final output, writing the expression for each gate. For the example circuit in Figure 4–18, the Boolean expression is determined in the following three steps:

A combinational logic circuit can be described by a Boolean equation.

1. The expression for the left-most AND gate with inputs C and D is CD.
2. The output of the left-most AND gate is one of the inputs to the OR gate and B is the other input. Therefore, the expression for the OR gate is $B + CD$.
3. The output of the OR gate is one of the inputs to the right-most AND gate and A is the other input. Therefore, the expression for this AND gate is $A(B + CD)$, which is the final output expression for the entire circuit.



◄ **FIGURE 4-18** A combinational logic circuit showing the development of the Boolean expression for the output.

Constructing a Truth Table for a Logic Circuit

Once the Boolean expression for a given logic circuit has been determined, a truth table that shows the output for all possible values of the input variables can be developed. The procedure requires that you evaluate the Boolean expression for all possible combinations of values for the input variables. In the case of the circuit in Figure 4–18, there are four input variables (A, B, C, and D) and therefore sixteen ($2^4 = 16$) combinations of values are possible.

A combinational logic circuit can be described by a truth table.

Evaluating the Expression

To evaluate the expression $A(B + CD)$, first find the values of the variables that make the expression equal to 1, using the rules for Boolean addition and multiplication. In this case, the expression equals 1 only if $A = 1$ and $B + CD = 1$ because

$$A(B + CD) = 1 \cdot 1 = 1$$

Now determine when the $B + CD$ term equals 1. The term $B + CD = 1$ if either $B = 1$ or $CD = 1$ or if both B and CD equal 1 because

$$B + CD = 1 + 0 = 1$$
$$B + CD = 0 + 1 = 1$$
$$B + CD = 1 + 1 = 1$$

The term $CD = 1$ only if $C = 1$ and $D = 1$.

To summarize, the expression $A(B + CD) = 1$ when $A = 1$ and $B = 1$ regardless of the values of C and D or when $A = 1$ and $C = 1$ and $D = 1$ regardless of the value of B. The expression $A(B + CD) = 0$ for all other value combinations of the variables.

Putting the Results in Truth Table Format

The first step is to list the sixteen input variable combinations of 1s and 0s in a binary sequence as shown in Table 4–5. Next, place a 1 in the output column for each combination of input variables that was determined in the evaluation. Finally, place a 0 in the output column for all other combinations of input variables. These results are shown in the truth table in Table 4–5.

TABLE 4-5
Truth table for the logic circuit in Figure 4–18.

Inputs				Output
A	B	C	D	$A(B + CD)$
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

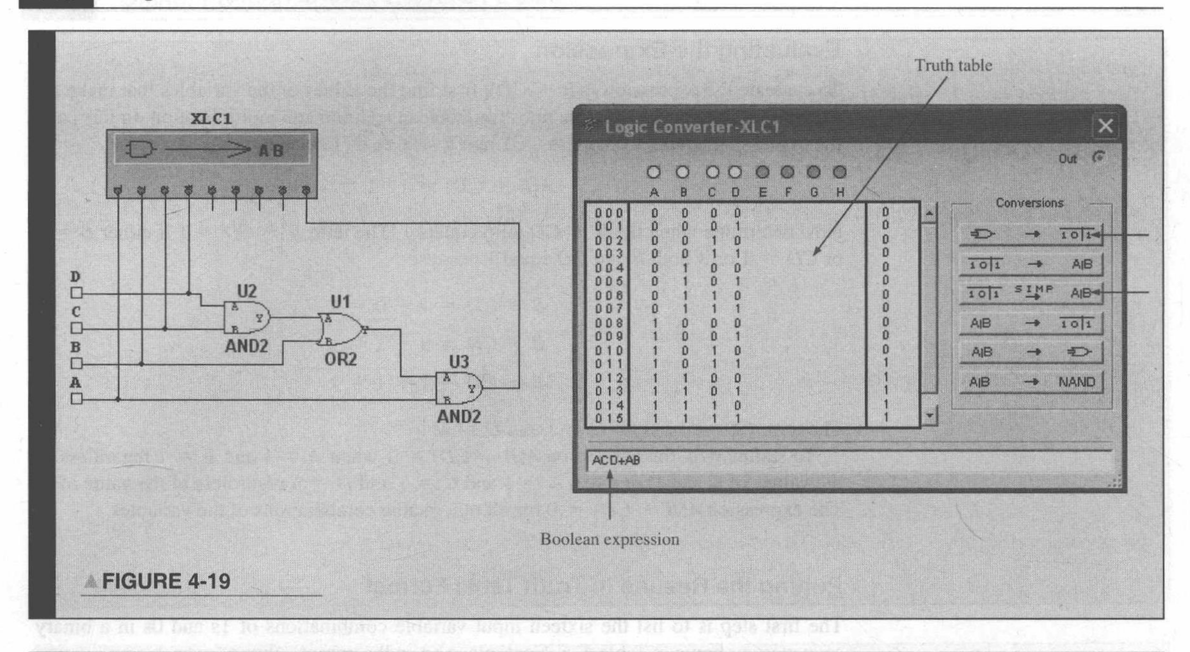
EXAMPLE 4-8

Use Multisim to generate the truth table for the logic circuit in Figure 4–18.

Solution

Construct the circuit in Multisim and connect the Multisim Logic Converter to the inputs and output, as shown in Figure 4–19. Click on the [→ 1 0|1] conversion bar, and the truth table appears in the display as shown.

You can also generate the simplified Boolean expression from the truth table by clicking on [1 0|1 SIMP A|B].



▲ FIGURE 4-19

SECTION 4-4 CHECKUP

1. Replace the AND gates with OR gates and the OR gate with an AND gate in Figure 4–18. Determine the Boolean expression for the output.
2. Construct a truth table for the circuit in Question 1.

4-5 Logic Simplification Using Boolean Algebra

A simplified Boolean expression uses the fewest gates possible to implement a given expression. Examples 4–9 through 4–12 illustrate Boolean simplification.

EXAMPLE 4-9

Using Boolean algebra techniques, simplify this expression:

$$AB + A(B + C) + B(B + C)$$

Solution

The following is not necessarily the only approach.

Step 1: Apply the distributive law to the second and third terms in the expression, as follows:

$$AB + AB + AC + BB + BC$$

Step 2: Apply rule 7 ($BB = B$) to the fourth term.

$$AB + AB + AC + B + BC$$

Step 3: Apply rule 5 ($AB + AB = AB$) to the first two terms.

$$AB + AC + B + BC$$

Step 4: Apply rule 10 ($B + BC = B$) to the last two terms.

$$AB + AC + B$$

Step 5: Apply rule 10 ($AB + B = B$) to the first and third terms.

$$B + AC$$

At this point the expression is simplified as much as possible. Once you gain experience in applying Boolean algebra, you can often combine many individual steps.

Related Problem

Simplify the Boolean expression $A\overline{B} + A(\overline{B + C}) + B(\overline{B + C})$.

Simplification means fewer gates for the same function.

Figure 4–20 shows that the simplification process in Example 4–9 has significantly reduced the number of logic gates required to implement the expression. Part (a) shows that five gates are required to implement the expression in its original form; however, only two gates are needed for the simplified expression, shown in part (b). It is important to realize that these two gate circuits are equivalent. That is, for any combination of levels on the A, B, and C inputs, you get the same output from either circuit.

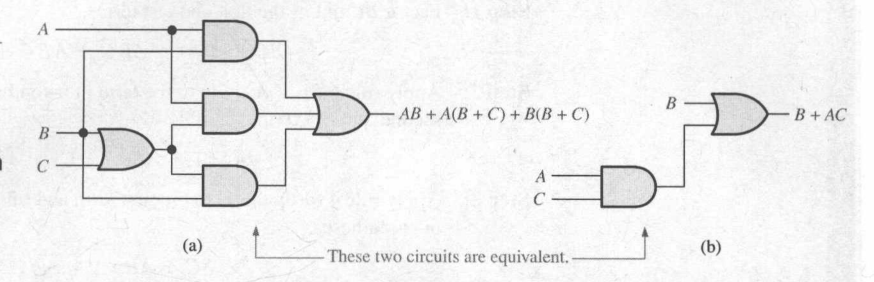


FIGURE 4-20 Gate circuits for Example 4–9. Open file F04-20 to verify equivalency.

MultiSim

EXAMPLE 4-10

Simplify the following Boolean expression:

$$[A\overline{B}(C + BD) + \overline{A}\overline{B}]C$$

Note that brackets and parentheses mean the same thing: the term inside is multiplied (ANDed) with the term outside.

Solution

Step 1: Apply the distributive law to the terms within the brackets.

$$(A\overline{B}C + A\overline{B}BD + \overline{A}\overline{B})C$$

Step 2: Apply rule 8 ($\overline{B}B = 0$) to the second term within the parentheses.

$$(A\overline{B}C + A \cdot 0 \cdot D + \overline{A}\overline{B})C$$

Step 3: Apply rule 3 ($A \cdot 0 \cdot D = 0$) to the second term within the parentheses.

$$(A\overline{B}C + 0 + \overline{A}\overline{B})C$$

Step 4: Apply rule 1 (drop the 0) within the parentheses.

$$(A\overline{B}C + \overline{A}\overline{B})C$$

Step 5: Apply the distributive law.

$$A\overline{B}CC + \overline{A}\overline{B}C$$

Step 6: Apply rule 7 ($CC = C$) to the first term.

$$A\overline{B}C + \overline{A}\overline{B}C$$

Step 7: Factor out $\overline{B}C$.

$$\overline{B}C(A + \overline{A})$$

Step 8: Apply rule 6 ($A + \overline{A} = 1$).

$$\overline{B}C \cdot 1$$

Step 9: Apply rule 4 (drop the 1).

$$\overline{B}C$$

Related Problem

Simplify the Boolean expression $[AB(C + \overline{BD}) + \overline{AB}]CD$.

EXAMPLE 4-11

Simplify the following Boolean expression:

$$\overline{A}BC + A\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,\overline{C} + A\overline{B}C + ABC$$

Solution

Step 1: Factor BC out of the first and last terms.

$$BC(\overline{A} + A) + A\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,\overline{C} + A\overline{B}C$$

Step 2: Apply rule 6 ($\overline{A} + A = 1$) to the term in parentheses, and factor $A\overline{B}$ from the second and last terms.

$$BC \cdot 1 + A\overline{B}(\overline{C} + C) + \overline{A}\,\overline{B}\,\overline{C}$$

Step 3: Apply rule 4 (drop the 1) to the first term and rule 6 ($\overline{C} + C = 1$) to the term in parentheses.

$$BC + A\overline{B} \cdot 1 + \overline{A}\,\overline{B}\,\overline{C}$$

Step 4: Apply rule 4 (drop the 1) to the second term.

$$BC + A\overline{B} + \overline{A}\,\overline{B}\,\overline{C}$$

Step 5: Factor $\overline{B}$ from the second and third terms.

$$BC + \overline{B}(A + \overline{A}\,\overline{C})$$

Step 6: Apply rule 11 ($A + \overline{A}\,\overline{C} = A + \overline{C}$) to the term in parentheses.

$$BC + \overline{B}(A + \overline{C})$$

Step 7: Use the distributive and commutative laws to get the following expression:

$$BC + A\overline{B} + \overline{B}\,\overline{C}$$

Related Problem

Simplify the Boolean expression $AB\overline{C} + \overline{A}\,\overline{B}C + \overline{A}BC + \overline{A}\,\overline{B}\,\overline{C}$.

EXAMPLE 4-12

Simplify the following Boolean expression:

$$\overline{AB + AC} + \overline{A}\,\overline{B}C$$

Solution

Step 1: Apply DeMorgan's theorem to the first term.

$$(\overline{AB})(\overline{AC}) + \overline{A}\,\overline{B}C$$

Step 2: Apply DeMorgan's theorem to each term in parentheses.

$$(\overline{A} + \overline{B})(\overline{A} + \overline{C}) + \overline{A}\,\overline{B}C$$

Step 3: Apply the distributive law to the two terms in parentheses.

$$\overline{A}\,\overline{A} + \overline{A}\,\overline{C} + \overline{A}\,\overline{B} + \overline{B}\,\overline{C} + \overline{A}\,\overline{B}C$$

Step 4: Apply rule 7 ($\overline{A}\,\overline{A} = \overline{A}$) to the first term, and apply rule 10 [$\overline{A}\,\overline{B} + \overline{A}\,\overline{B}C = \overline{A}\,\overline{B}(1 + C) = \overline{A}\,\overline{B}$] to the third and last terms.

$$\overline{A} + \overline{A}\,\overline{C} + \overline{A}\,\overline{B} + \overline{B}\,\overline{C}$$

Step 5: Apply rule 10 $[\overline{A} + \overline{A}\overline{C} = \overline{A}(1 + \overline{C}) = \overline{A}]$ to the first and second terms.

$$\overline{A} + \overline{A}\overline{B} + \overline{B}\overline{C}$$

Step 6: Apply rule 10 $[\overline{A} + \overline{A}\overline{B} = \overline{A}(1 + \overline{B}) = \overline{A}]$ to the first and second terms.

$$\overline{A} + \overline{B}\overline{C}$$

Related Problem

Simplify the Boolean expression $\overline{AB} + \overline{AC} + \overline{A}\overline{B}\overline{C}$.

EXAMPLE 4-13

Use Multisim to perform the logic simplification shown in Figure 4–20.

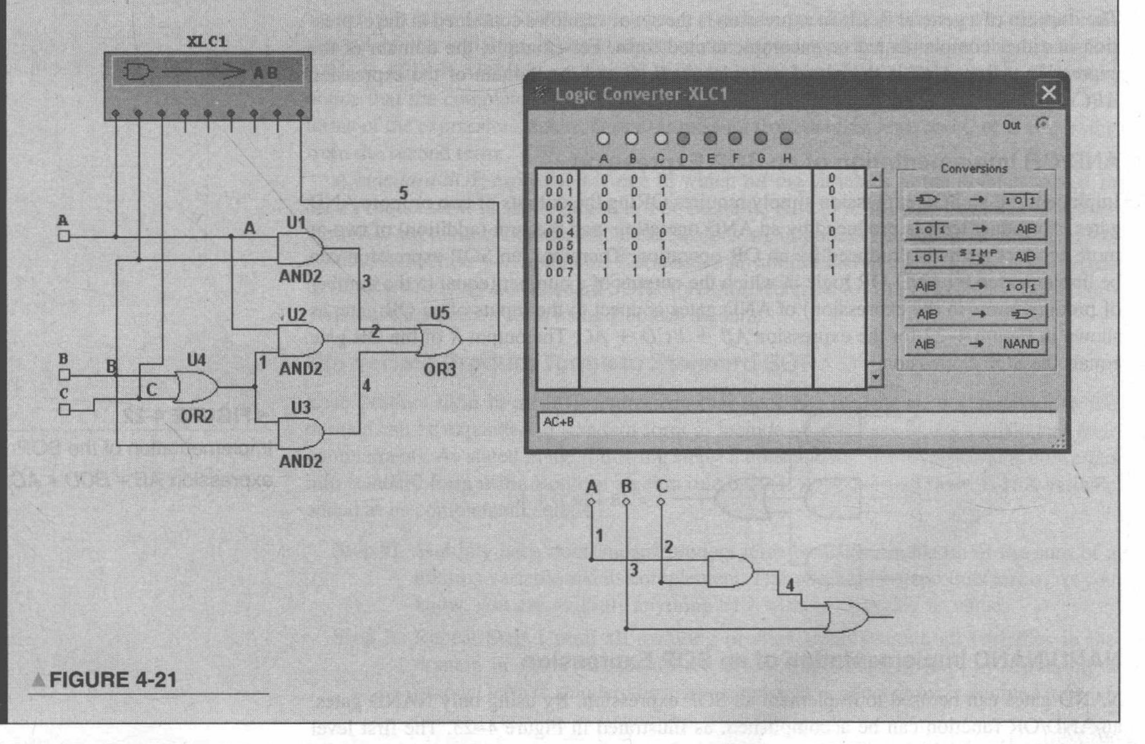
Solution

Step 1: Connect the Multisim Logic Converter to the circuit as shown in Figure 4–21.

Step 2: Generate the truth table by clicking on [⇒D → 1011].

Step 3: Generate the simplified Boolean expression by clicking on [1011 SIMP A|B].

Step 4: Generate the simplified logic circuit by clicking on [A|B → ⇒D].



▲**FIGURE 4-21**

SECTION 4-5 CHECKUP

1. Simplify the following Boolean expressions:

   (a) $A + AB + A\overline{B}C$ (b) $(\overline{A} + B)C + ABC$ (c) $A\overline{B}C(BD + CDE) + A\overline{C}$

2. Implement each expression in Question 1 as originally stated with the appropriate logic gates. Then implement the simplified expression, and compare the number of gates.

4-6 Standard Forms of Boolean Expressions

The Sum-of-Products (SOP) Form

An SOP expression can be implemented with one OR gate and two or more AND gates.

A product term was defined in Section 4–1 as a term consisting of the product (Boolean multiplication) of literals (variables or their complements). When two or more product terms are summed by Boolean addition, the resulting expression is a **sum-of-products (SOP)**. Some examples are

$$AB + ABC$$
$$ABC + CDE + \overline{B}C\overline{D}$$
$$\overline{A}B + \overline{A}B\overline{C} + AC$$

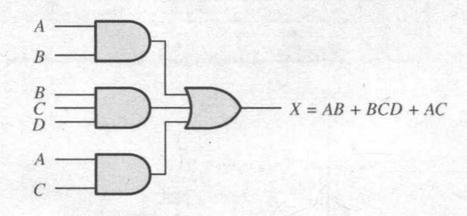
Also, an SOP expression can contain a single-variable term, as in $A + \overline{A}\,\overline{B}C + BC\overline{D}$. Refer to the simplification examples in the last section, and you will see that each of the final expressions was either a single product term or in SOP form. In an SOP expression, a single overbar cannot extend over more than one variable; however, more than one variable in a term can have an overbar. For example, an SOP expression can have the term $\overline{A}\,\overline{B}\,\overline{C}$ but not $\overline{ABC}$.

Domain of a Boolean Expression

The **domain** of a general Boolean expression is the set of variables contained in the expression in either complemented or uncomplemented form. For example, the domain of the expression $\overline{A}B + A\overline{B}C$ is the set of variables A, B, C and the domain of the expression $AB\overline{C} + C\overline{D}E + \overline{B}C\overline{D}$ is the set of variables A, B, C, D, E.

AND/OR Implementation of an SOP Expression

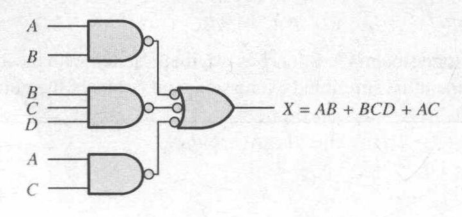
Implementing an SOP expression simply requires ORing the outputs of two or more AND gates. A product term is produced by an AND operation, and the sum (addition) of two or more product terms is produced by an OR operation. Therefore, an SOP expression can be implemented by AND-OR logic in which the outputs of a number (equal to the number of product terms in the expression) of AND gates connect to the inputs of an OR gate, as shown in Figure 4–22 for the expression $AB + BCD + AC$. The output X of the OR gate equals the SOP expression.



◀ **FIGURE 4-22**
Implementation of the SOP expression $AB + BCD + AC$.

NAND/NAND Implementation of an SOP Expression

NAND gates can be used to implement an SOP expression. By using only NAND gates, an AND/OR function can be accomplished, as illustrated in Figure 4–23. The first level of NAND gates feed into a NAND gate that acts as a negative-OR gate. The NAND and negative-OR inversions cancel and the result is effectively an AND/OR circuit.



◀ **FIGURE 4-23**
This NAND/NAND implementation is equivalent to the AND/OR in Figure 4–22.

Conversion of a General Expression to SOP Form

Any logic expression can be changed into SOP form by applying Boolean algebra techniques. For example, the expression $A(B + CD)$ can be converted to SOP form by applying the distributive law:

$$A(B + CD) = AB + ACD$$

EXAMPLE 4-14

Convert each of the following Boolean expressions to SOP form:

(a) $AB + B(CD + EF)$ **(b)** $(A + B)(B + C + D)$ **(c)** $\overline{(\overline{A + B}) + C}$

Solution

(a) $AB + B(CD + EF) = AB + BCD + BEF$

(b) $(A + B)(B + C + D) = AB + AC + AD + BB + BC + BD$

(c) $\overline{(\overline{A + B}) + C} = \overline{(\overline{A + B})}\overline{C} = (A + B)\overline{C} = A\overline{C} + B\overline{C}$

Related Problem

Convert $\overline{A}B\overline{C} + (A + \overline{B})(B + \overline{C} + A\overline{B})$ to SOP form.

The Standard SOP Form

So far, you have seen SOP expressions in which some of the product terms do not contain all of the variables in the domain of the expression. For example, the expression $\overline{A}B\overline{C} + A\overline{B}D + \overline{A}B\overline{C}D$ has a domain made up of the variables A, B, C, and D. However, notice that the complete set of variables in the domain is not represented in the first two terms of the expression; that is, D or $\overline{D}$ is missing from the first term and C or $\overline{C}$ is missing from the second term.

A *standard SOP expression* is one in which *all* the variables in the domain appear in each product term in the expression. For example, $A\overline{B}CD + \overline{A}\,\overline{B}C\overline{D} + AB\overline{C}\,\overline{D}$ is a standard SOP expression. Standard SOP expressions are important in constructing truth tables, covered in Section 4–7, and in the Karnaugh map simplification method, which is covered in Section 4–8. Any nonstandard SOP expression (referred to simply as SOP) can be converted to the standard form using Boolean algebra.

Converting Product Terms to Standard SOP

Each product term in an SOP expression that does not contain all the variables in the domain can be expanded to standard form to include all variables in the domain and their complements. As stated in the following steps, a nonstandard SOP expression is converted into standard form using Boolean algebra rule 6 ($A + \overline{A} = 1$) from Table 4–1: A variable added to its complement equals 1.

Step 1: Multiply each nonstandard product term by a term made up of the sum of a missing variable and its complement. This results in two product terms. As you know, you can multiply anything by 1 without changing its value.

Step 2: Repeat Step 1 until all resulting product terms contain all variables in the domain in either complemented or uncomplemented form. In converting a product term to standard form, the number of product terms is doubled for each missing variable, as Example 4–15 shows.

EXAMPLE 4-15

Convert the following Boolean expression into standard SOP form:

$$A\overline{B}C + \overline{A}\,\overline{B} + AB\overline{C}D$$

Solution

The domain of this SOP expression is A, B, C, D. Take one term at a time. The first term, $A\overline{B}C$, is missing variable D or $\overline{D}$, so multiply the first term by $D + \overline{D}$ as follows:

$$A\overline{B}C = A\overline{B}C(D + \overline{D}) = A\overline{B}CD + A\overline{B}C\overline{D}$$

In this case, two standard product terms are the result.

The second term, $\overline{A}\,\overline{B}$, is missing variables C or $\overline{C}$ and D or $\overline{D}$, so first multiply the second term by $C + \overline{C}$ as follows:

$$\overline{A}\,\overline{B} = \overline{A}\,\overline{B}(C + \overline{C}) = \overline{A}\,\overline{B}C + \overline{A}\,\overline{B}\,\overline{C}$$

The two resulting terms are missing variable D or $\overline{D}$, so multiply both terms by $D + \overline{D}$ as follows:

$$\overline{A}\,\overline{B} = \overline{A}\,\overline{B}C + \overline{A}\,\overline{B}\,\overline{C} = \overline{A}\,\overline{B}C(D + \overline{D}) + \overline{A}\,\overline{B}\,\overline{C}(D + \overline{D})$$
$$= \overline{A}\,\overline{B}CD + \overline{A}\,\overline{B}C\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$$

In this case, four standard product terms are the result.

The third term, $ABC\overline{D}$, is already in standard form. The complete standard SOP form of the original expression is as follows:

$$A\overline{B}C + \overline{A}\,\overline{B} + ABC\overline{D} = A\overline{B}CD + A\overline{B}C\overline{D} + \overline{A}\,\overline{B}CD + \overline{A}\,\overline{B}C\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + ABC\overline{D}$$

Related Problem

Convert the expression $W\overline{X}Y + \overline{X}Y\overline{Z} + WX\overline{Y}$ to standard SOP form.

Binary Representation of a Standard Product Term

A standard product term is equal to 1 for only one combination of variable values. For example, the product term $A\overline{B}C\overline{D}$ is equal to 1 when $A = 1, B = 0, C = 1, D = 0$, as shown below, and is 0 for all other combinations of values for the variables.

$$A\overline{B}C\overline{D} = 1 \cdot \overline{0} \cdot 1 \cdot \overline{0} = 1 \cdot 1 \cdot 1 \cdot 1 = 1$$

In this case, the product term has a binary value of 1010 (decimal ten).

Remember, a product term is implemented with an AND gate whose output is 1 only if each of its inputs is 1. Inverters are used to produce the complements of the variables as required.

An SOP expression is equal to 1 only if one or more of the product terms in the expression is equal to 1.

EXAMPLE 4-16

Determine the binary values for which the following standard SOP expression is equal to 1:

$$ABCD + A\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$$

Solution

The term $ABCD$ is equal to 1 when $A = 1, B = 1, C = 1$, and $D = 1$.

$$ABCD = 1 \cdot 1 \cdot 1 \cdot 1 = 1$$

The term $A\overline{B}\,\overline{C}D$ is equal to 1 when $A = 1, B = 0, C = 0$, and $D = 1$.

$$A\overline{B}\,\overline{C}D = 1 \cdot \overline{0} \cdot \overline{0} \cdot 1 = 1 \cdot 1 \cdot 1 \cdot 1 = 1$$

The term $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$ is equal to 1 when $A = 0, B = 0, C = 0$, and $D = 0$.

$$\overline{A}\,\overline{B}\,\overline{C}\,\overline{D} = \overline{0} \cdot \overline{0} \cdot \overline{0} \cdot \overline{0} = 1 \cdot 1 \cdot 1 \cdot 1 = 1$$

The SOP expression equals 1 when any or all of the three product terms is 1.

Related Problem

Determine the binary values for which the following SOP expression is equal to 1:

$$\overline{X}YZ + X\overline{Y}Z + XY\overline{Z} + \overline{X}Y\overline{Z} + XYZ$$

Is this a standard SOP expression?

The Product-of-Sums (POS) Form

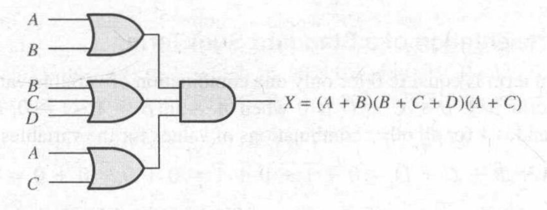
A sum term was defined in Section 4–1 as a term consisting of the sum (Boolean addition) of literals (variables or their complements). When two or more sum terms are multiplied, the resulting expression is a **product-of-sums (POS)**. Some examples are

$$(\overline{A} + B)(A + \overline{B} + C)$$
$$(\overline{A} + \overline{B} + \overline{C})(C + \overline{D} + E)(\overline{B} + C + D)$$
$$(A + B)(A + \overline{B} + C)(\overline{A} + C)$$

A POS expression can contain a single-variable term, as in $\overline{A}(A + \overline{B} + C)(\overline{B} + \overline{C} + D)$. In a POS expression, a single overbar cannot extend over more than one variable; however, more than one variable in a term can have an overbar. For example, a POS expression can have the term $\overline{A} + \overline{B} + \overline{C}$ but not $\overline{A + B + C}$.

Implementation of a POS Expression

Implementing a POS expression simply requires ANDing the outputs of two or more OR gates. A sum term is produced by an OR operation, and the product of two or more sum terms is produced by an AND operation. Therefore, a POS expression can be implemented by logic in which the outputs of a number (equal to the number of sum terms in the expression) of OR gates connect to the inputs of an AND gate, as Figure 4–24 shows for the expression $(A + B)(B + C + D)(A + C)$. The output X of the AND gate equals the POS expression.



▶ FIGURE 4-24
Implementation of the POS expression $(A + B)(B + C + D)(A + C)$.

The Standard POS Form

So far, you have seen POS expressions in which some of the sum terms do not contain all of the variables in the domain of the expression. For example, the expression

$$(A + \overline{B} + C)(A + B + \overline{D})(A + \overline{B} + \overline{C} + D)$$

has a domain made up of the variables A, B, C, and D. Notice that the complete set of variables in the domain is not represented in the first two terms of the expression; that is, D or $\overline{D}$ is missing from the first term and C or $\overline{C}$ is missing from the second term.

A *standard POS expression* is one in which *all* the variables in the domain appear in each sum term in the expression. For example,

$$(\overline{A} + \overline{B} + \overline{C} + \overline{D})(A + \overline{B} + C + D)(A + B + \overline{C} + D)$$

is a standard POS expression. Any nonstandard POS expression (referred to simply as POS) can be converted to the standard form using Boolean algebra.

Converting a Sum Term to Standard POS

Each sum term in a POS expression that does not contain all the variables in the domain can be expanded to standard form to include all variables in the domain and their complements. As stated in the following steps, a nonstandard POS expression is converted into standard form using Boolean algebra rule 8 ($A \cdot \overline{A} = 0$) from Table 4–1: A variable multiplied by its complement equals 0.

Step 1: Add to each nonstandard product term a term made up of the product of the missing variable and its complement. This results in two sum terms. As you know, you can add 0 to anything without changing its value.

Step 2: Apply rule 12 from Table 4–1: $A + BC = (A + B)(A + C)$

Step 3: Repeat Step 1 until all resulting sum terms contain all variables in the domain in either complemented or uncomplemented form.

EXAMPLE 4-17

Convert the following Boolean expression into standard POS form:

$$(A + \overline{B} + C)(\overline{B} + C + \overline{D})(A + \overline{B} + \overline{C} + D)$$

Solution

The domain of this POS expression is A, B, C, D. Take one term at a time. The first term, $A + \overline{B} + C$, is missing variable D or $\overline{D}$, so add $D\overline{D}$ and apply rule 12 as follows:

$$A + \overline{B} + C = A + \overline{B} + C + D\overline{D} = (A + \overline{B} + C + D)(A + \overline{B} + C + \overline{D})$$

The second term, $\overline{B} + C + \overline{D}$, is missing variable A or $\overline{A}$, so add $A\overline{A}$ and apply rule 12 as follows:

$$\overline{B} + C + \overline{D} = \overline{B} + C + \overline{D} + A\overline{A} = (A + \overline{B} + C + \overline{D})(\overline{A} + \overline{B} + C + \overline{D})$$

The third term, $A + \overline{B} + \overline{C} + D$, is already in standard form. The standard POS form of the original expression is as follows:

$$(A + \overline{B} + C)(\overline{B} + C + \overline{D})(A + \overline{B} + \overline{C} + D) =$$
$$(A + \overline{B} + C + D)(A + \overline{B} + C + \overline{D})(A + \overline{B} + C + \overline{D})(\overline{A} + \overline{B} + C + \overline{D})(A + \overline{B} + \overline{C} + D)$$

Related Problem

Convert the expression $(A + \overline{B})(B + C)$ to standard POS form.

Binary Representation of a Standard Sum Term

A standard sum term is equal to 0 for only one combination of variable values. For example, the sum term $A + \overline{B} + C + \overline{D}$ is 0 when $A = 0$, $B = 1$, $C = 0$, and $D = 1$, as shown below, and is 1 for all other combinations of values for the variables.

$$A + \overline{B} + C + \overline{D} = 0 + \overline{1} + 0 + \overline{1} = 0 + 0 + 0 + 0 = 0$$

In this case, the sum term has a binary value of 0101 (decimal 5). Remember, a sum term is implemented with an OR gate whose output is 0 only if each of its inputs is 0. Inverters are used to produce the complements of the variables as required.

A POS expression is equal to 0 only if one or more of the sum terms in the expression is equal to 0.

EXAMPLE 4-18

Determine the binary values of the variables for which the following standard POS expression is equal to 0:

$$(A + B + C + D)(A + \overline{B} + \overline{C} + D)(\overline{A} + \overline{B} + \overline{C} + \overline{D})$$

Solution

The term $A + B + C + D$ is equal to 0 when $A = 0$, $B = 0$, $C = 0$, and $D = 0$.

$$A + B + C + D = 0 + 0 + 0 + 0 = 0$$

The term $A + \overline{B} + \overline{C} + D$ is equal to 0 when $A = 0$, $B = 1$, $C = 1$, and $D = 0$.

$$A + \overline{B} + \overline{C} + D = 0 + \overline{1} + \overline{1} + 0 = 0 + 0 + 0 + 0 = 0$$

The term $\overline{A} + \overline{B} + \overline{C} + \overline{D}$ is equal to 0 when $A = 1$, $B = 1$, $C = 1$, and $D = 1$.

$$\overline{A} + \overline{B} + \overline{C} + \overline{D} = \overline{1} + \overline{1} + \overline{1} + \overline{1} = 0 + 0 + 0 + 0 = 0$$

The POS expression equals 0 when any of the three sum terms equals 0.

Related Problem

Determine the binary values for which the following POS expression is equal to 0:

$$(X + \overline{Y} + Z)(\overline{X} + Y + Z)(X + Y + \overline{Z})(\overline{X} + \overline{Y} + \overline{Z})(X + \overline{Y} + \overline{Z})$$

Is this a standard POS expression?

Converting Standard SOP to Standard POS

The binary values of the product terms in a given standard SOP expression are not present in the equivalent standard POS expression. Also, the binary values that are not represented in the SOP expression are present in the equivalent POS expression. Therefore, to convert from standard SOP to standard POS, the following steps are taken:

Step 1: Evaluate each product term in the SOP expression. That is, determine the binary numbers that represent the product terms.

Step 2: Determine all of the binary numbers not included in the evaluation in Step 1.

Step 3: Write the equivalent sum term for each binary number from Step 2 and express in POS form.

Using a similar procedure, you can go from POS to SOP.

EXAMPLE 4-19

Convert the following SOP expression to an equivalent POS expression:

$$\overline{A}\,\overline{B}\,\overline{C} + \overline{A}B\overline{C} + \overline{A}BC + A\overline{B}C + ABC$$

Solution

The evaluation is as follows:

$$000 + 010 + 011 + 101 + 111$$

Since there are three variables in the domain of this expression, there are a total of eight (2^3) possible combinations. The SOP expression contains five of these combinations, so the POS must contain the other three which are 001, 100, and 110. Remember, these are the binary values that make the sum term 0. The equivalent POS expression is

$$(A + B + \overline{C})(\overline{A} + B + C)(\overline{A} + \overline{B} + C)$$

Related Problem

Verify that the SOP and POS expressions in this example are equivalent by substituting binary values into each.

SECTION 4-6 CHECKUP

1. Identify each of the following expressions as SOP, standard SOP, POS, or standard POS:
   (a) $AB + \overline{A}BD + \overline{A}C\overline{D}$ (b) $(A + \overline{B} + C)(A + B + \overline{C})$
   (c) $\overline{A}BC + AB\overline{C}$ (d) $(A + \overline{C})(A + B)$
2. Convert each SOP expression in Question 1 to standard form.
3. Convert each POS expression in Question 1 to standard form.

4-7 Boolean Expressions and Truth Tables

Converting SOP Expressions to Truth Table Format

Recall from Section 4–6 that an SOP expression is equal to 1 only if at least one of the product terms is equal to 1. A truth table is simply a list of the possible combinations of input variable values and the corresponding output values (1 or 0). For an expression with a domain of two variables, there are four different combinations of those variables ($2^2 = 4$). For an expression with a domain of three variables, there are eight different combinations of those variables ($2^3 = 8$). For an expression with a domain of four variables, there are sixteen different combinations of those variables ($2^4 = 16$), and so on.

The first step in constructing a truth table is to list all possible combinations of binary values of the variables in the expression. Next, convert the SOP expression to standard

form if it is not already. Finally, place a 1 in the output column (X) for each binary value that makes the standard SOP expression a 1 and place a 0 for all the remaining binary values. This procedure is illustrated in Example 4–20.

EXAMPLE 4-20

Develop a truth table for the standard SOP expression $\overline{A}\,\overline{B}C + A\overline{B}\,\overline{C} + ABC$.

Solution

There are three variables in the domain, so there are eight possible combinations of binary values of the variables as listed in the left three columns of Table 4–6. The binary values that make the product terms in the expressions equal to 1 are $\overline{A}\,\overline{B}C$: 001; $A\overline{B}\,\overline{C}$: 100; and ABC: 111. For each of these binary values, place a 1 in the output column as shown in the table. For each of the remaining binary combinations, place a 0 in the output column.

TABLE 4-6

Inputs			Output	
A	B	C	X	Product Term
0	0	0	0	
0	0	1	1	$\overline{A}\,\overline{B}C$
0	1	0	0	
0	1	1	0	
1	0	0	1	$A\overline{B}\,\overline{C}$
1	0	1	0	
1	1	0	0	
1	1	1	1	ABC

Related Problem

Create a truth table for the standard SOP expression $\overline{A}B\overline{C} + A\overline{B}C$.

Converting POS Expressions to Truth Table Format

Recall that a POS expression is equal to 0 only if at least one of the sum terms is equal to 0. To construct a truth table from a POS expression, list all the possible combinations of binary values of the variables just as was done for the SOP expression. Next, convert the POS expression to standard form if it is not already. Finally, place a 0 in the output column (X) for each binary value that makes the expression a 0 and place a 1 for all the remaining binary values. This procedure is illustrated in Example 4–21.

EXAMPLE 4-21

Determine the truth table for the following standard POS expression:

$$(A + B + C)(A + \overline{B} + C)(A + \overline{B} + \overline{C})(\overline{A} + B + \overline{C})(\overline{A} + \overline{B} + C)$$

Solution

There are three variables in the domain and the eight possible binary values are listed in the left three columns of Table 4–7. The binary values that make the sum terms in the expression equal to 0 are $A + B + C$: 000; $A + \overline{B} + C$: 010; $A + \overline{B} + \overline{C}$: 011; $\overline{A} + B + \overline{C}$: 101; and $\overline{A} + \overline{B} + C$: 110. For each of these binary values, place a 0 in the output column as shown in the table. For each of the remaining binary combinations, place a 1 in the output column.

TABLE 4-7

Inputs			Output	
A	B	C	X	Sum Term
0	0	0	0	$(A + B + C)$
0	0	1	1	
0	1	0	0	$(A + \overline{B} + C)$
0	1	1	0	$(A + \overline{B} + \overline{C})$
1	0	0	1	
1	0	1	0	$(\overline{A} + B + \overline{C})$
1	1	0	0	$(\overline{A} + \overline{B} + C)$
1	1	1	1	

Notice that the truth table in this example is the same as the one in Example 4–20. This means that the SOP expression in the previous example and the POS expression in this example are equivalent.

Related Problem

Develop a truth table for the following standard POS expression:

$$(A + \overline{B} + C)(A + B + \overline{C})(\overline{A} + \overline{B} + \overline{C})$$

Determining Standard Expressions from a Truth Table

To determine the standard SOP expression represented by a truth table, list the binary values of the input variables for which the output is 1. Convert each binary value to the corresponding product term by replacing each 1 with the corresponding variable and each 0 with the corresponding variable complement. For example, the binary value 1010 is converted to a product term as follows:

$$1010 \longrightarrow A\overline{B}C\overline{D}$$

If you substitute, you can see that the product term is 1:

$$A\overline{B}C\overline{D} = 1 \cdot \overline{0} \cdot 1 \cdot \overline{0} = 1 \cdot 1 \cdot 1 \cdot 1 = 1$$

To determine the standard POS expression represented by a truth table, list the binary values for which the output is 0. Convert each binary value to the corresponding sum term by replacing each 1 with the corresponding variable complement and each 0 with the corresponding variable. For example, the binary value 1001 is converted to a sum term as follows:

$$1001 \longrightarrow \overline{A} + B + C + \overline{D}$$

If you substitute, you can see that the sum term is 0:

$$\overline{A} + B + C + \overline{D} = \overline{1} + 0 + 0 + \overline{1} = 0 + 0 + 0 + 0 = 0$$

EXAMPLE 4-22

From the truth table in Table 4–8, determine the standard SOP expression and the equivalent standard POS expression.

TABLE 4-8

Inputs			Output
A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Solution

There are four 1s in the output column and the corresponding binary values are 011, 100, 110, and 111. Convert these binary values to product terms as follows:

$$011 \longrightarrow \overline{A}BC$$
$$100 \longrightarrow A\overline{B}\,\overline{C}$$
$$110 \longrightarrow AB\overline{C}$$
$$111 \longrightarrow ABC$$

The resulting standard SOP expression for the output X is

$$X = \overline{A}BC + A\overline{B}\,\overline{C} + AB\overline{C} + ABC$$

For the POS expression, the output is 0 for binary values 000, 001, 010, and 101. Convert these binary values to sum terms as follows:

$$000 \longrightarrow A + B + C$$
$$001 \longrightarrow A + B + \overline{C}$$
$$010 \longrightarrow A + \overline{B} + C$$
$$101 \longrightarrow \overline{A} + B + \overline{C}$$

The resulting standard POS expression for the output X is

$$X = (A + B + C)(A + B + \overline{C})(A + \overline{B} + C)(\overline{A} + B + \overline{C})$$

Related Problem

By substitution of binary values, show that the SOP and the POS expressions derived in this example are equivalent; that is, for any binary value each SOP and POS term should either both be 1 or both be 0, depending on the binary value.

SECTION 4-7 CHECKUP

1. If a certain Boolean expression has a domain of five variables, how many binary values will be in its truth table?
2. In a certain truth table, the output is a 1 for the binary value 0110. Convert this binary value to the corresponding product term using variables W, X, Y, and Z.
3. In a certain truth table, the output is a 0 for the binary value 1100. Convert this binary value to the corresponding sum term using variables W, X, Y, and Z.

4-8 The Karnaugh Map

The purpose of a Karnaugh map is to simplify a Boolean expression.

A **Karnaugh map** is similar to a truth table because it presents all of the possible values of input variables and the resulting output for each value. Instead of being organized into columns and rows like a truth table, the Karnaugh map is an array of **cells** in which each cell represents a binary value of the input variables. The cells are arranged in a way so that simplification of a given expression is simply a matter of properly grouping the cells. Karnaugh maps can be used for expressions with two, three, four, and five variables, but we will discuss only 3-variable and 4-variable situations to illustrate the principles. *A discussion of 5-variable Karnaugh maps is available on the website.*

The number of cells in a Karnaugh map, as well as the number of rows in a truth table, is equal to the total number of possible input variable combinations. For three variables, the number of cells is $2^3 = 8$. For four variables, the number of cells is $2^4 = 16$.

The 3-Variable Karnaugh Map

The 3-variable Karnaugh map is an array of eight cells, as shown in Figure 4–25(a). In this case, A, B, and C are used for the variables although other letters could be used. Binary values of A and B are along the left side (notice the sequence) and the values of C are across

the top. The value of a given cell is the binary values of A and B at the left in the same row combined with the value of C at the top in the same column. For example, the cell in the upper left corner has a binary value of 000 and the cell in the lower right corner has a binary value of 101. Figure 4–25(b) shows the standard product terms that are represented by each cell in the Karnaugh map.

► **FIGURE 4-25**
A 3-variable Karnaugh map showing Boolean product terms for each cell.

(a)

AB \ C	0	1
00		
01		
11		
10		

(b)

AB \ C	0	1
00	$\overline{A}\,\overline{B}\,\overline{C}$	$\overline{A}\,\overline{B}C$
01	$\overline{A}B\overline{C}$	$\overline{A}BC$
11	$AB\overline{C}$	ABC
10	$A\overline{B}\,\overline{C}$	$A\overline{B}C$

The 4-Variable Karnaugh Map

The 4-variable Karnaugh map is an array of sixteen cells, as shown in Figure 4–26(a). Binary values of A and B are along the left side and the values of C and D are across the top. The value of a given cell is the binary values of A and B at the left in the same row combined with the binary values of C and D at the top in the same column. For example, the cell in the upper right corner has a binary value of 0010 and the cell in the lower right corner has a binary value of 1010. Figure 4–26(b) shows the standard product terms that are represented by each cell in the 4-variable Karnaugh map.

Cell Adjacency

Cells that differ by only one variable are adjacent.

Cells with values that differ by more than one variable are not adjacent.

The cells in a Karnaugh map are arranged so that there is only a single-variable change between adjacent cells. **Adjacency** is defined by a single-variable change. In the 3-variable map the 010 cell is adjacent to the 000 cell, the 011 cell, and the 110 cell. The 010 cell is not adjacent to the 001 cell, the 111 cell, the 100 cell, or the 101 cell.

Physically, each cell is adjacent to the cells that are immediately next to it on any of its four sides. A cell is not adjacent to the cells that diagonally touch any of its corners. Also, the cells in the top row are adjacent to the corresponding cells in the bottom row and

► **FIGURE 4-26**
A 4-variable Karnaugh map.

(a)

AB \ CD	00	01	11	10
00				
01				
11				
10				

(b)

AB \ CD	00	01	11	10
00	$\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$	$\overline{A}\,\overline{B}\,\overline{C}D$	$\overline{A}\,\overline{B}CD$	$\overline{A}\,\overline{B}C\overline{D}$
01	$\overline{A}B\overline{C}\,\overline{D}$	$\overline{A}B\overline{C}D$	$\overline{A}BCD$	$\overline{A}BC\overline{D}$
11	$AB\overline{C}\,\overline{D}$	$AB\overline{C}D$	$ABCD$	$ABC\overline{D}$
10	$A\overline{B}\,\overline{C}\,\overline{D}$	$A\overline{B}\,\overline{C}D$	$A\overline{B}CD$	$A\overline{B}C\overline{D}$

the cells in the outer left column are adjacent to the corresponding cells in the outer right column. This is called "wrap-around" adjacency because you can think of the map as wrapping around from top to bottom to form a cylinder or from left to right to form a cylinder. Figure 4–27 illustrates the cell adjacencies with a 4-variable map, although the same rules for adjacency apply to Karnaugh maps with any number of cells.

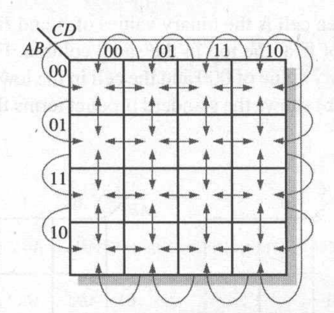


FIGURE 4-27 Adjacent cells on a Karnaugh map are those that differ by only one variable. Arrows point between adjacent cells.

The Quine-McCluskey Method

Minimizing Boolean functions using Karnaugh maps is practical only for up to four or five variables. Also, the Karnaugh map method does not lend itself to be automated in the form of a computer program.

The Quine-McCluskey method is more practical for logic simplification of functions with more than four or five variables. It also has the advantage of being easily implemented with a computer or programmable calculator.

The Quine-McCluskey method is functionally similar to Karnaugh mapping, but the tabular form makes it more efficient for use in computer algorithms, and it also gives a way to check that the minimal form of a Boolean function has been reached. This method is sometimes referred to as the *tabulation method.* An introduction to the Quine-McCluskey method is provided in Section 4–11.

Espresso Algorithm

Although the Quine-McCluskey method is well suited to be implemented in a computer program and can handle more variables than the Karnaugh map method, the result is still far from efficient in terms of processing time and memory usage. Adding a variable to the function will roughly double both of these parameters because the truth table length increases exponentially with the number of variables. Functions with a large number of variables have to be minimized with other methods such as the Espresso logic minimizer, which has become the de facto world standard. *An Espresso algorithm tutorial is available on the website.*

Compared to the other methods, Espresso is essentially more efficient in terms of reducing memory usage and computation time by several orders of magnitude. There is essentially no restrictions to the number of variables, output functions, and product terms of a combinational logic function. In general, tens of variables with tens of output functions can be handled by Espresso.

The Espresso algorithm has been incorporated as a standard logic function minimization step in most logic synthesis tools for programmable logic devices. For implementing a function in multilevel logic, the minimization result is optimized by factorization and mapped onto the available basic logic cells in the target device, such as an FPGA (Field-Programmable Gate Array).

SECTION 4-8 CHECKUP

1. In a 3-variable Karnaugh map, what is the binary value for the cell in each of the following locations:
   (a) upper left corner (b) lower right corner
   (c) lower left corner (d) upper right corner
2. What is the standard product term for each cell in Question 1 for variables *X*, *Y*, and *Z*?
3. Repeat Question 1 for a 4-variable map.
4. Repeat Question 2 for a 4-variable map using variables *W*, *X*, *Y*, and *Z*.

4-9 Karnaugh Map SOP Minimization

Mapping a Standard SOP Expression

For an SOP expression in standard form, a 1 is placed on the Karnaugh map for each product term in the expression. Each 1 is placed in a cell corresponding to the value of a product term. For example, for the product term $A\overline{B}C$, a 1 goes in the 101 cell on a 3-variable map.

When an SOP expression is completely mapped, there will be a number of 1s on the Karnaugh map equal to the number of product terms in the standard SOP expression. The cells that do not have a 1 are the cells for which the expression is 0. Usually, when working with SOP expressions, the 0s are left off the map. The following steps and the illustration in Figure 4–28 show the mapping process.

Step 1: Determine the binary value of each product term in the standard SOP expression. After some practice, you can usually do the evaluation of terms mentally.

Step 2: As each product term is evaluated, place a 1 on the Karnaugh map in the cell having the same value as the product term.

$$\overline{A}\overline{B}\overline{C} + \overline{A}\overline{B}C + AB\overline{C} + A\overline{B}\overline{C}$$
$$000 \quad 001 \quad 110 \quad 100$$

AB \ C	0	1
00	1	1
01		
11	1	
10	1	

FIGURE 4-28
Example of mapping a standard SOP expression.

EXAMPLE 4-23

Map the following standard SOP expression on a Karnaugh map:

$$\overline{A}\overline{B}C + \overline{A}B\overline{C} + AB\overline{C} + ABC$$

Solution

Evaluate the expression as shown below. Place a 1 on the 3-variable Karnaugh map in Figure 4–29 for each standard product term in the expression.

$$\overline{A}\overline{B}C + \overline{A}B\overline{C} + AB\overline{C} + ABC$$
$$001 \quad 010 \quad 110 \quad 111$$

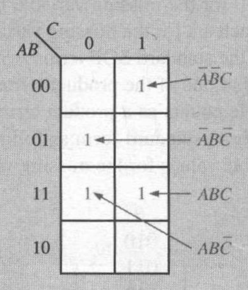


FIGURE 4-29

Related Problem

Map the standard SOP expression $\overline{A}BC + A\overline{B}C + A\overline{B}\overline{C}$ on a Karnaugh map.

EXAMPLE 4-24

Map the following standard SOP expression on a Karnaugh map:

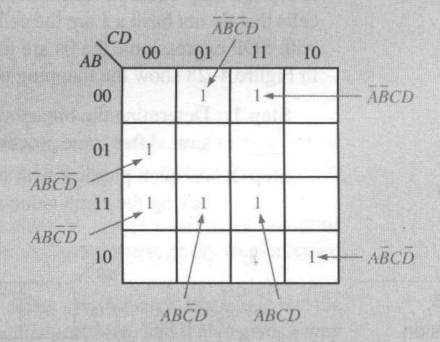
$$\bar{A}\bar{B}CD + \bar{A}B\bar{C}\bar{D} + AB\bar{C}D + ABCD + AB\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D + A\bar{B}C\bar{D}$$

Solution

Evaluate the expression as shown below. Place a 1 on the 4-variable Karnaugh map in Figure 4–30 for each standard product term in the expression.

$\bar{A}\bar{B}CD$	$\bar{A}B\bar{C}\bar{D}$	$AB\bar{C}D$	$ABCD$	$AB\bar{C}\bar{D}$	$\bar{A}\bar{B}\bar{C}D$	$A\bar{B}C\bar{D}$
0011	0100	1101	1111	1100	0001	1010

▶ FIGURE 4-30



Related Problem

Map the following standard SOP expression on a Karnaugh map:

$$\bar{A}BC\bar{D} + ABC\bar{D} + AB\bar{C}\bar{D} + ABCD$$

Mapping a Nonstandard SOP Expression

A Boolean expression must first be in standard form before you use a Karnaugh map. If an expression is not in standard form, then it must be converted to standard form by the procedure covered in Section 4–6 or by numerical expansion. Since an expression should be evaluated before mapping anyway, numerical expansion is probably the most efficient approach.

Numerical Expansion of a Nonstandard Product Term

Recall that a nonstandard product term has one or more missing variables. For example, assume that one of the product terms in a certain 3-variable SOP expression is $A\bar{B}$. This term can be expanded numerically to standard form as follows. First, write the binary value of the two variables and attach a 0 for the missing variable $\bar{C}$: 100. Next, write the binary value of the two variables and attach a 1 for the missing variable C: 101. The two resulting binary numbers are the values of the standard SOP terms $A\bar{B}\bar{C}$ and $A\bar{B}C$.

As another example, assume that one of the product terms in a 3-variable expression is B (remember that a single variable counts as a product term in an SOP expression). This term can be expanded numerically to standard form as follows. Write the binary value of the variable; then attach all possible values for the missing variables A and C as follows:

B
010
011
110
111

The four resulting binary numbers are the values of the standard SOP terms $\bar{A}B\bar{C}$, $\bar{A}BC$, $AB\bar{C}$, and ABC.

EXAMPLE 4-25

Map the following SOP expression on a Karnaugh map: $\overline{A} + A\overline{B} + AB\overline{C}$.

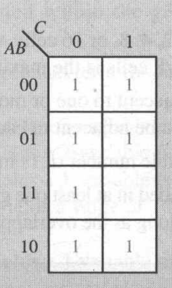
Solution

The SOP expression is obviously not in standard form because each product term does not have three variables. The first term is missing two variables, the second term is missing one variable, and the third term is standard. First expand the terms numerically as follows:

$\overline{A}$	$+ A\overline{B}$	$+ AB\overline{C}$
000	100	110
001	101	
010		
011		

Map each of the resulting binary values by placing a 1 in the appropriate cell of the 3-variable Karnaugh map in Figure 4–31.

FIGURE 4-31



Related Problem

Map the SOP expression $BC + \overline{A}\,\overline{C}$ on a Karnaugh map.

EXAMPLE 4-26

Map the following SOP expression on a Karnaugh map:

$$\overline{B}\,\overline{C} + A\overline{B} + AB\overline{C} + A\overline{B}C\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + A\overline{B}CD$$

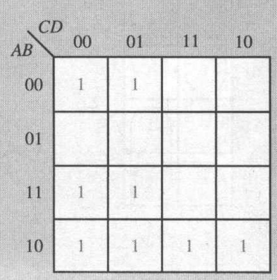
Solution

The SOP expression is obviously not in standard form because each product term does not have four variables. The first and second terms are both missing two variables, the third term is missing one variable, and the rest of the terms are standard. First expand the terms by including all combinations of the missing variables numerically as follows:

$\overline{B}\,\overline{C}$	$+ A\overline{B}$	$+ AB\overline{C}$	$+ A\overline{B}C\overline{D}$	$+ \overline{A}\,\overline{B}\,\overline{C}D$	$+ A\overline{B}CD$
0000	1000	1100	1010	0001	1011
0001	1001	1101			
1000	1010				
1001	1011				

Map each of the resulting binary values by placing a 1 in the appropriate cell of the 4-variable Karnaugh map in Figure 4–32. Notice that some of the values in the expanded expression are redundant.

FIGURE 4-32



Related Problem

Map the expression $A + \overline{C}D + AC\overline{D} + \overline{A}BC\overline{D}$ on a Karnaugh map.

Karnaugh Map Simplification of SOP Expressions

The process that results in an expression containing the fewest possible terms with the fewest possible variables is called **minimization**. After an SOP expression has been mapped, a minimum SOP expression is obtained by grouping the 1s and determining the minimum SOP expression from the map.

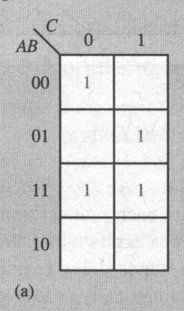
Grouping the 1s

You can group 1s on the Karnaugh map according to the following rules by enclosing those adjacent cells containing 1s. The goal is to maximize the size of the groups and to minimize the number of groups.

1. A group must contain either 1, 2, 4, 8, or 16 cells, which are all powers of two. In the case of a 3-variable map, $2^3 = 8$ cells is the maximum group.
2. Each cell in a group must be adjacent to one or more cells in that same group, but all cells in the group do not have to be adjacent to each other.
3. Always include the largest possible number of 1s in a group in accordance with rule 1.
4. Each 1 on the map must be included in at least one group. The 1s already in a group can be included in another group as long as the overlapping groups include noncommon 1s.

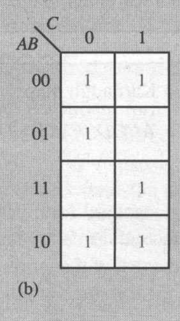
EXAMPLE 4-27

Group the 1s in each of the Karnaugh maps in Figure 4–33.



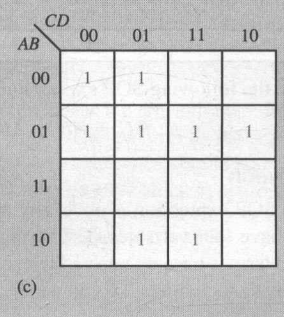
AB \ C	0	1
00	1	
01		1
11	1	1
10		

(a)



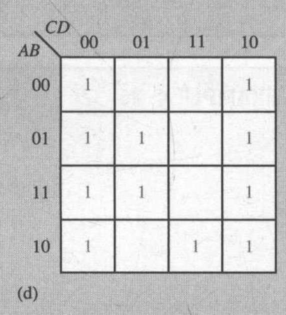
AB \ C	0	1
00	1	1
01	1	
11		1
10	1	1

(b)



AB \ CD	00	01	11	10
00	1	1		
01	1	1	1	1
11				
10		1	1	

(c)



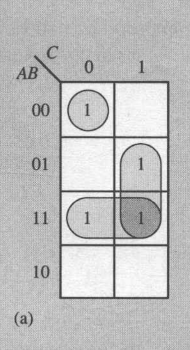
AB \ CD	00	01	11	10
00	1			1
01	1	1		1
11	1	1		1
10	1		1	1

(d)

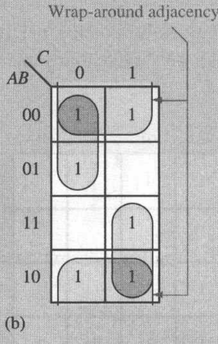
▲ **FIGURE 4-33**

Solution

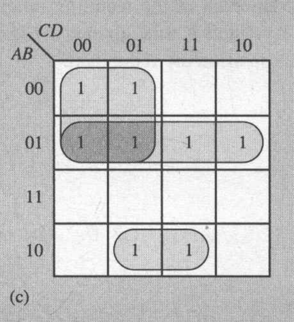
The groupings are shown in Figure 4–34. In some cases, there may be more than one way to group the 1s to form maximum groupings.



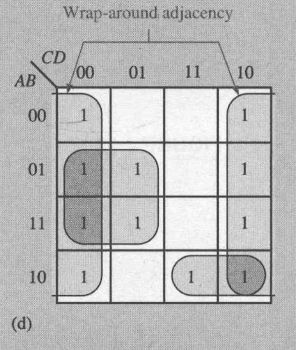
(a)



(b)



(c)



(d)

▲ **FIGURE 4-34**

Related Problem

Determine if there are other ways to group the 1s in Figure 4–34 to obtain a minimum number of maximum groupings.

Determining the Minimum SOP Expression from the Map

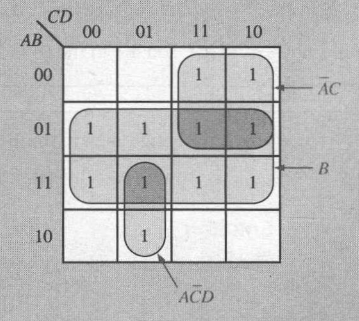
When all the 1s representing the standard product terms in an expression are properly mapped and grouped, the process of determining the resulting minimum SOP expression begins. The following rules are applied to find the minimum product terms and the minimum SOP expression:

1. Group the cells that have 1s. Each group of cells containing 1s creates one product term composed of all variables that occur in only one form (either uncomplemented or complemented) within the group. Variables that occur both uncomplemented and complemented within the group are eliminated. These are called *contradictory variables.*
2. Determine the minimum product term for each group.
   - **(a)** For a 3-variable map:
     - **(1)** A 1-cell group yields a 3-variable product term
     - **(2)** A 2-cell group yields a 2-variable product term
     - **(3)** A 4-cell group yields a 1-variable term
     - **(4)** An 8-cell group yields a value of 1 for the expression
   - **(b)** For a 4-variable map:
     - **(1)** A 1-cell group yields a 4-variable product term
     - **(2)** A 2-cell group yields a 3-variable product term
     - **(3)** A 4-cell group yields a 2-variable product term
     - **(4)** An 8-cell group yields a 1-variable term
     - **(5)** A 16-cell group yields a value of 1 for the expression
3. When all the minimum product terms are derived from the Karnaugh map, they are summed to form the minimum SOP expression.

EXAMPLE 4-28

Determine the product terms for the Karnaugh map in Figure 4–35 and write the resulting minimum SOP expression.

► **FIGURE 4-35**



Solution

Eliminate variables that are in a grouping in both complemented and uncomplemented forms. In Figure 4–35, the product term for the 8-cell group is B because the cells within that group contain both A and $\overline{A}$, C and $\overline{C}$, and D and $\overline{D}$, which are eliminated. The 4-cell group contains B, $\overline{B}$, D, and $\overline{D}$, leaving the variables $\overline{A}$ and C, which form the product term $\overline{A}C$. The 2-cell group contains B and $\overline{B}$, leaving variables A, $\overline{C}$, and D which form the product term $A\overline{C}D$. Notice how overlapping is used to maximize the size of the groups. The resulting minimum SOP expression is the sum of these product terms:

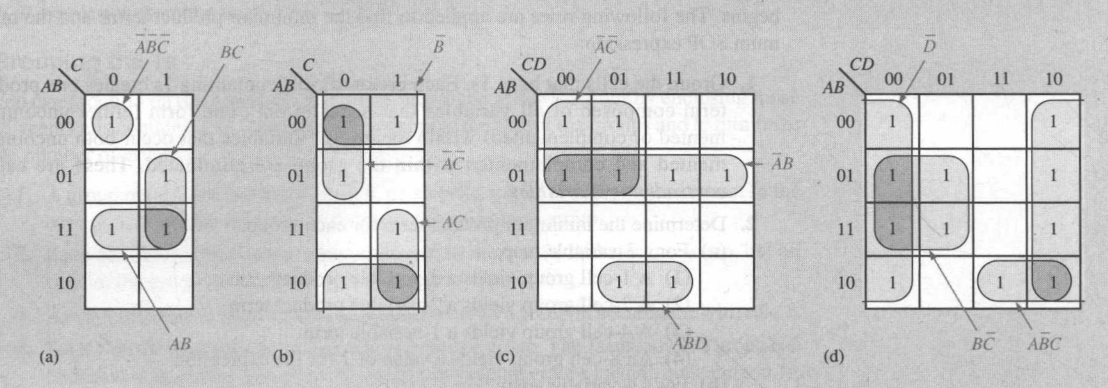
$$B + \overline{A}C + A\overline{C}D$$

Related Problem

For the Karnaugh map in Figure 4–35, add a 1 in the lower right cell (1010) and determine the resulting SOP expression.

EXAMPLE 4-29

Determine the product terms for each of the Karnaugh maps in Figure 4–36 and write the resulting minimum SOP expression.



▲ FIGURE 4-36

Solution

The resulting minimum product term for each group is shown in Figure 4–36. The minimum SOP expressions for each of the Karnaugh maps in the figure are

(a) $AB + BC + \overline{A}\,\overline{B}\,\overline{C}$

(b) $\overline{B} + \overline{A}\,\overline{C} + AC$

(c) $\overline{A}B + \overline{A}\,\overline{C} + A\overline{B}D$

(d) $\overline{D} + A\overline{B}C + B\overline{C}$

Related Problem

For the Karnaugh map in Figure 4–36(d), add a 1 in the 0111 cell and determine the resulting SOP expression.

EXAMPLE 4-30

Use a Karnaugh map to minimize the following standard SOP expression:

$$A\overline{B}C + \overline{A}BC + \overline{A}\,\overline{B}C + \overline{A}\,\overline{B}\,\overline{C} + A\overline{B}\,\overline{C}$$

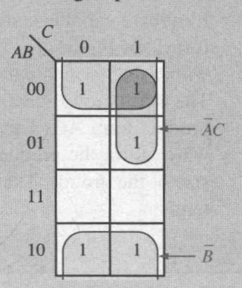
Solution

The binary values of the expression are

$$101 + 011 + 001 + 000 + 100$$

Map the standard SOP expression and group the cells as shown in Figure 4–37.

► FIGURE 4-37



Notice the "wrap around" 4-cell group that includes the top row and the bottom row of 1s. The remaining 1 is absorbed in an overlapping group of two cells. The group of four 1s produces a single variable term, $\overline{B}$. This is determined by observing that within the group, $\overline{B}$ is the only variable that does not change from cell to cell. The group of two 1s produces a 2-variable term $\overline{A}C$. This is determined by observing that within the group, $\overline{A}$ and C do not change from one cell to the next. The product term for each group is shown. The resulting minimum SOP expression is

$$\overline{B} + \overline{A}C$$

Keep in mind that this minimum expression is equivalent to the original standard expression.

Related Problem

Use a Karnaugh map to simplify the following standard SOP expression:

$$X\overline{Y}Z + XY\overline{Z} + \overline{X}YZ + \overline{X}Y\overline{Z} + X\overline{Y}\,\overline{Z} + XYZ$$

EXAMPLE 4-31

Use a Karnaugh map to minimize the following SOP expression:

$$\overline{B}\,\overline{C}\,\overline{D} + \overline{A}B\overline{C}\,\overline{D} + AB\overline{C}\,\overline{D} + \overline{A}\,\overline{B}CD + A\overline{B}CD + \overline{A}\,\overline{B}C\overline{D} + \overline{A}BC\overline{D} + ABC\overline{D} + A\overline{B}C\overline{D}$$

Solution

The first term $\overline{B}\,\overline{C}\,\overline{D}$ must be expanded into $A\overline{B}\,\overline{C}\,\overline{D}$ and $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$ to get the standard SOP expression, which is then mapped; the cells are grouped as shown in Figure 4–38.

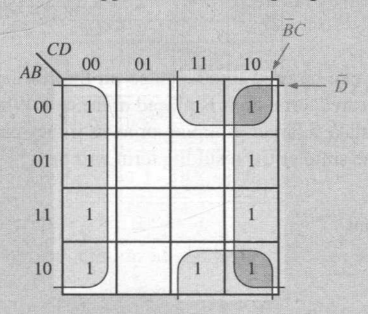


FIGURE 4-38

Notice that both groups exhibit "wrap around" adjacency. The group of eight is formed because the cells in the outer columns are adjacent. The group of four is formed to pick up the remaining two 1s because the top and bottom cells are adjacent. The product term for each group is shown. The resulting minimum SOP expression is

$$\overline{D} + \overline{B}C$$

Keep in mind that this minimum expression is equivalent to the original standard expression.

Related Problem

Use a Karnaugh map to simplify the following SOP expression:

$$\overline{W}\,\overline{X}\,\overline{Y}\,\overline{Z} + W\overline{X}YZ + W\overline{X}\,\overline{Y}Z + \overline{W}YZ + W\overline{X}\,\overline{Y}\,\overline{Z}$$

Mapping Directly from a Truth Table

You have seen how to map a Boolean expression; now you will learn how to go directly from a truth table to a Karnaugh map. Recall that a truth table gives the output of a Boolean expression for all possible input variable combinations. An example of a Boolean expression and its truth table representation is shown in Figure 4–39. Notice in the truth table that the output X is 1 for four different input variable combinations. The 1s in the output column of the truth table are mapped directly onto a Karnaugh map into the cells corresponding to

the values of the associated input variable combinations, as shown in Figure 4–39. In the figure you can see that the Boolean expression, the truth table, and the Karnaugh map are simply different ways to represent a logic function.

"Don't Care" Conditions

Sometimes a situation arises in which some input variable combinations are not allowed. For example, recall that in the BCD code covered in Chapter 2, there are six invalid combinations: 1010, 1011, 1100, 1101, 1110, and 1111. Since these unallowed states will never occur in an application involving the BCD code, they can be treated as **"don't care"** terms with respect to their effect on the output. That is, for these "don't care" terms either a 1 or a 0 may be assigned to the output; it really does not matter since they will never occur.

$X = \overline{A}\overline{B}\overline{C} + A\overline{B}\overline{C} + AB\overline{C} + ABC$

A	B	C	X
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

AB \ C	0	1
00	1	
01		
11	1	1
10	1	

FIGURE 4-39 Example of mapping directly from a truth table to a Karnaugh map.

The "don't care" terms can be used to advantage on the Karnaugh map. Figure 4–40 shows that for each "don't care" term, an X is placed in the cell. When grouping the 1s, the Xs can be treated as 1s to make a larger grouping or as 0s if they cannot be used to advantage. The larger a group, the simpler the resulting term will be.

A	B	C	D	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

Don't cares

(a) Truth table

AB \ CD	00	01	11	10
00				
01			1	
11	X	X	X	X
10	1	1	X	X

$\overline{A}BCD$, BCD, $A\overline{B}\overline{C}$, A

(b) Without "don't cares" $Y = A\overline{B}\overline{C} + \overline{A}BCD$
With "don't cares" $Y = A + BCD$

FIGURE 4-40 Example of the use of "don't care" conditions to simplify an expression.

The truth table in Figure 4–40(a) describes a logic function that has a 1 output only when the BCD code for 7, 8, or 9 is present on the inputs. If the "don't cares" are used as 1s, the resulting expression for the function is $A + BCD$, as indicated in part (b). If the "don't cares" are not used as 1s, the resulting expression is $A\overline{B}\,\overline{C} + \overline{A}BCD$; so you can see the advantage of using "don't care" terms to get the simplest expression.

EXAMPLE 4-32

In a 7-segment display, each of the seven segments is activated for various digits. For example, segment a is activated for the digits 0, 2, 3, 5, 6, 7, 8, and 9, as illustrated in Figure 4–41. Since each digit can be represented by a BCD code, derive an SOP expression for segment a using the variables $ABCD$ and then minimize the expression using a Karnaugh map.

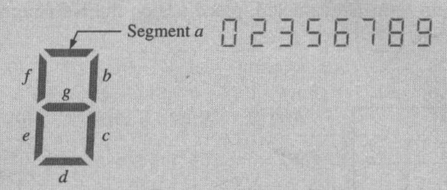


FIGURE 4-41
7-segment display.

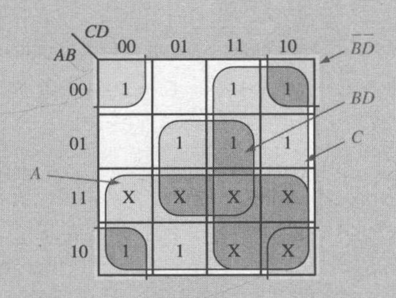
Solution

The expression for segment a is

$$a = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}C\overline{D} + \overline{A}\,\overline{B}CD + \overline{A}B\overline{C}D + \overline{A}BC\overline{D} + \overline{A}BCD + A\overline{B}\,\overline{C}\,\overline{D} + A\overline{B}\,\overline{C}D$$

Each term in the expression represents one of the digits in which segment a is used. The Karnaugh map minimization is shown in Figure 4–42. X's (don't cares) are entered for those states that do not occur in the BCD code.

FIGURE 4-42



From the Karnaugh map, the minimized expression for segment a is

$$a = A + C + BD + \overline{B}\,\overline{D}$$

Related Problem

Draw the logic diagram for the segment-a logic.

SECTION 4-9 CHECKUP

1. Lay out Karnaugh maps for three and four variables.
2. Group the 1s and write the simplified SOP expression for the Karnaugh map in Figure 4–29.
3. Write the original standard SOP expressions for each of the Karnaugh maps in Figure 4–36.

4-10 Karnaugh Map POS Minimization

Mapping a Standard POS Expression

For a POS expression in standard form, a 0 is placed on the Karnaugh map for each sum term in the expression. Each 0 is placed in a cell corresponding to the value of a sum term. For example, for the sum term $A + \overline{B} + C$, a 0 goes in the 010 cell on a 3-variable map.

When a POS expression is completely mapped, there will be a number of 0s on the Karnaugh map equal to the number of sum terms in the standard POS expression. The cells

that do not have a 0 are the cells for which the expression is 1. Usually, when working with POS expressions, the 1s are left off. The following steps and the illustration in Figure 4–43 show the mapping process.

Step 1: Determine the binary value of each sum term in the standard POS expression. This is the binary value that makes the term equal to 0.

Step 2: As each sum term is evaluated, place a 0 on the Karnaugh map in the corresponding cell.

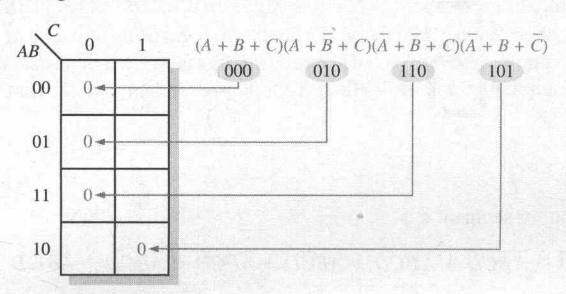


FIGURE 4-43 Example of mapping a standard POS expression.

EXAMPLE 4-33

Map the following standard POS expression on a Karnaugh map:

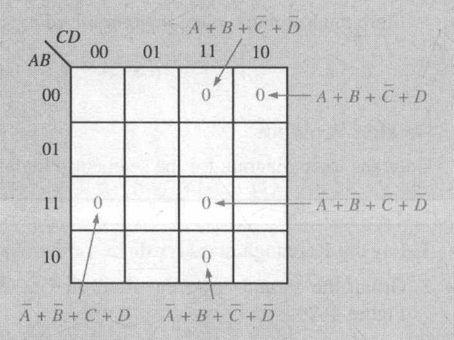
$$(\overline{A} + \overline{B} + C + D)(\overline{A} + B + \overline{C} + \overline{D})(A + B + \overline{C} + D)(\overline{A} + \overline{B} + \overline{C} + \overline{D})(A + B + \overline{C} + \overline{D})$$

Solution

Evaluate the expression as shown below and place a 0 on the 4-variable Karnaugh map in Figure 4–44 for each standard sum term in the expression.

$$\underset{1100}{(\overline{A} + \overline{B} + C + D)}\underset{1011}{(\overline{A} + B + \overline{C} + \overline{D})}\underset{0010}{(A + B + \overline{C} + D)}\underset{1111}{(\overline{A} + \overline{B} + \overline{C} + \overline{D})}\underset{0011}{(A + B + \overline{C} + \overline{D})}$$

FIGURE 4-44



Related Problem

Map the following standard POS expression on a Karnaugh map:

$$(A + \overline{B} + \overline{C} + D)(A + B + C + \overline{D})(A + B + C + D)(\overline{A} + B + \overline{C} + D)$$

Karnaugh Map Simplification of POS Expressions

The process for minimizing a POS expression is basically the same as for an SOP expression except that you group 0s to produce minimum sum terms instead of grouping 1s to produce minimum product terms. The rules for grouping the 0s are the same as those for grouping the 1s that you learned in Section 4–9.

EXAMPLE 4-34

Use a Karnaugh map to minimize the following standard POS expression:

$$(A + B + C)(A + B + \overline{C})(A + \overline{B} + C)(A + \overline{B} + \overline{C})(\overline{A} + \overline{B} + C)$$

Also, derive the equivalent SOP expression.

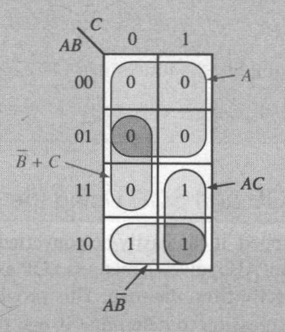
Solution

The combinations of binary values of the expression are

$$(0 + 0 + 0)(0 + 0 + 1)(0 + 1 + 0)(0 + 1 + 1)(1 + 1 + 0)$$

Map the standard POS expression and group the cells as shown in Figure 4–45.

FIGURE 4-45



Notice how the 0 in the 110 cell is included into a 2-cell group by utilizing the 0 in the 4-cell group. The sum term for each blue group is shown in the figure and the resulting minimum POS expression is

$$A(\overline{B} + C)$$

Keep in mind that this minimum POS expression is equivalent to the original standard POS expression.

Grouping the 1s as shown by the gray areas yields an SOP expression that is equivalent to grouping the 0s.

$$AC + A\overline{B} = A(\overline{B} + C)$$

Related Problem

Use a Karnaugh map to simplify the following standard POS expression:

$$(X + \overline{Y} + Z)(X + \overline{Y} + \overline{Z})(\overline{X} + \overline{Y} + Z)(\overline{X} + Y + Z)$$

EXAMPLE 4-35

Use a Karnaugh map to minimize the following POS expression:

$$(B + C + D)(A + B + \overline{C} + D)(\overline{A} + B + C + \overline{D})(A + \overline{B} + C + D)(\overline{A} + \overline{B} + C + D)$$

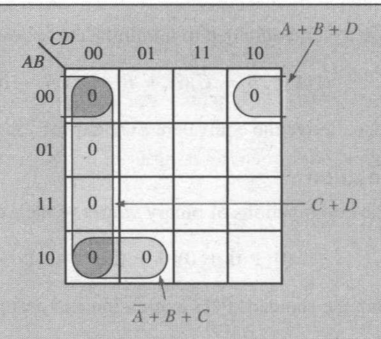
Solution

The first term must be expanded into $\overline{A} + B + C + D$ and $A + B + C + D$ to get a standard POS expression, which is then mapped; and the cells are grouped as shown in Figure 4–46. The sum term for each group is shown and the resulting minimum POS expression is

$$(C + D)(A + B + D)(\overline{A} + B + C)$$

Keep in mind that this minimum POS expression is equivalent to the original standard POS expression.

FIGURE 4-46



Related Problem

Use a Karnaugh map to simplify the following POS expression:

$$(W + \bar{X} + Y + \bar{Z})(W + X + Y + Z)(W + \bar{X} + \bar{Y} + Z)(\bar{W} + \bar{X} + Z)$$

Converting Between POS and SOP Using the Karnaugh Map

When a POS expression is mapped, it can easily be converted to the equivalent SOP form directly from the Karnaugh map. Also, given a mapped SOP expression, an equivalent POS expression can be derived directly from the map. This provides a good way to compare both minimum forms of an expression to determine if one of them can be implemented with fewer gates than the other.

For a POS expression, all the cells that do not contain 0s contain 1s, from which the SOP expression is derived. Likewise, for an SOP expression, all the cells that do not contain 1s contain 0s, from which the POS expression is derived. Example 4–36 illustrates this conversion.

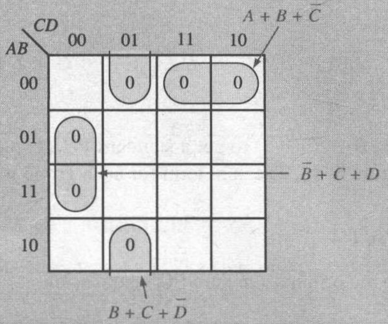
EXAMPLE 4-36

Using a Karnaugh map, convert the following standard POS expression into a minimum POS expression, a standard SOP expression, and a minimum SOP expression.

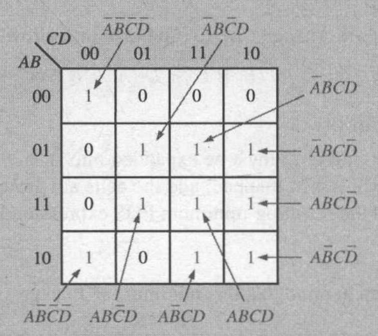
$$(\bar{A} + \bar{B} + C + D)(A + \bar{B} + C + D)(A + B + C + \bar{D})(A + B + \bar{C} + \bar{D})(\bar{A} + B + C + \bar{D})(A + B + \bar{C} + D)$$

Solution

The 0s for the standard POS expression are mapped and grouped to obtain the minimum POS expression in Figure 4–47(a). In Figure 4–47(b), 1s are added to the cells that do not contain 0s. From each cell containing a 1, a standard product term is obtained as indicated. These product terms form the standard SOP expression. In Figure 4–47(c), the 1s are grouped and a minimum SOP expression is obtained.

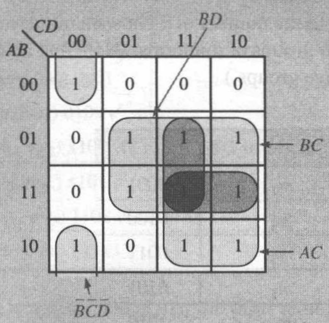


(a) Minimum POS: $(A + B + C)(\bar{B} + \bar{C} + D)(B + C + \bar{D})$



(b) Standard SOP:
$\bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}B\bar{C}D + \bar{A}BCD + \bar{A}BC\bar{D} + ABC\bar{D} + A\bar{B}C\bar{D} +$
$A\bar{B}\bar{C}\bar{D} + AB\bar{C}D + A\bar{B}CD + ABCD$

▶ FIGURE 4-47



(c) Minimum SOP: $AC + BC + BD + \overline{B}\overline{C}\overline{D}$

Related Problem

Use a Karnaugh map to convert the following expression to minimum SOP form:

$$(W + \overline{X} + Y + \overline{Z})(\overline{W} + X + \overline{Y} + \overline{Z})(\overline{W} + \overline{X} + \overline{Y} + Z)(\overline{W} + \overline{X} + \overline{Z})$$

SECTION 4-10 CHECKUP

1. What is the difference in mapping a POS expression and an SOP expression?
2. What is the standard sum term for a 0 in cell 1011?
3. What is the standard product term for a 1 in cell 0010?

4-11 The Quine-McCluskey Method

Unlike the Karnaugh mapping method, Quine-McCluskey lends itself to the computerized reduction of Boolean expressions, which is its principal use. For simple expressions, with up to four or perhaps even five variables, the Karnaugh map is easier for most people because it is a graphic method.

To apply the Quine-McCluskey method, first write the function in standard **minterm** (SOP) form. To illustrate, we will use the expression

$$X = \overline{A}\overline{B}\overline{C}D + \overline{A}\overline{B}CD + \overline{A}B\overline{C}\overline{D} + \overline{A}B\overline{C}D + A\overline{B}C\overline{D} + AB\overline{C}\overline{D} + AB\overline{C}D + ABCD$$

and represent it as binary numbers on the truth table shown in Table 4–9. The minterms that appear in the function are listed in the right column.

▶ TABLE 4-9

ABCD	*X*	Minterm
0000	0	
0001	1	m_1
0010	0	
0011	1	m_3
0100	1	m_4
0101	1	m_5
0110	0	
0111	0	
1000	0	
1001	0	
1010	1	m_{10}
1011	0	
1100	1	m_{12}
1101	1	m_{13}
1110	0	
1111	1	m_{15}

The second step in applying the Quine-McCluskey method is to arrange the minterms in the original expression in groups according to the number of 1s in each minterm, as shown in Table 4–10. In this example, there are four groups of minterms. (Note that if m_0 had been in the original expression, there would be five groups.)

TABLE 4-10

Number of 1s	Minterm	*ABCD*
1	m_1	0001
	m_4	0100
2	m_3	0011
	m_5	0101
	m_{10}	1010
	m_{12}	1100
3	m_{13}	1101
4	m_{15}	1111

Third, compare adjacent groups, looking to see if any minterms are the same in every position *except one.* If they are, place a check mark by those two minterms, as shown in Table 4–11. You should check each minterm against all others in the following group, but it is not necessary to check any groups that are not adjacent. In the column labeled *First Level,* you will have a list of the minterm names and the binary equivalent with an x as the placeholder for the literal that differs. In the example, minterm m_1 in Group 1 (0001) is identical to m_3 in Group 2 (0011) except for the *C* position, so place a check mark by these two minterms and enter 00x1 in the column labeled *First Level.* Minterm m_4 (0100) is identical to m_5 (0101) except for the *D* position, so check these two minterms and enter 010x in the last column. If a given term can be used more than once, it should be. In this case, notice that m_1 can be used again with m_5 in the second row with the x now placed in the *B* position.

TABLE 4-11

Number of 1s in Minterm	Minterm	ABCD	First Level
1	m_1	0001 ✓	(m_1, m_3) 00x1
	m_4	0100 ✓	(m_1, m_5) 0x01
2	m_3	0011 ✓	(m_4, m_5) 010x
	m_5	0101 ✓	(m_4, m_{12}) x100
	m_{10}	1010	(m_5, m_{13}) x101
	m_{12}	1100 ✓	(m_{12}, m_{13}) 110x
3	m_{13}	1101 ✓	(m_{13}, m_{15}) 11x1
4	m_{15}	1111 ✓	

In Table 4–11, minterm m_4 and minterm m_{12} are identical except for the *A* position. Both minterms are checked and x100 is entered in the *First Level* column. Follow this procedure for groups 2 and 3. In these groups, m_5 and m_{13} are combined and so are m_{12} and m_{13} (notice that m_{12} was previously used with m_4 and is used again). For groups 3 and 4, both m_{13} and m_{15} are added to the list in the *First Level* column.

In this example, minterm m_{10} does not have a check mark because no other minterm meets the requirement of being identical except for one position. This term is called an *essential prime implicant,* and it must be included in our final reduced expression.

The terms listed in the *First Level* have been used to form a reduced table (Table 4–12) with one less group than before. The number of 1s remaining in the *First Level* are counted and used to form three new groups.

Terms in the new groups are compared against terms in the adjacent group down. You need to compare these terms only if the x is in the same relative position in adjacent groups; otherwise go on. If the two expressions differ by exactly one position, a check mark is placed next to both terms as before and all of the minterms are listed in the Second Level list. As before, the one position that has changed is entered as an x in the *Second Level.*

▶ TABLE 4-12

First Level	Number of 1s in First Level	Second Level
(m_1, m_3) 00x1	1	$(m_4, m_5, m_{12}, m_{13})$ x10x
(m_1, m_5) 0x01		$(m_4, m_5, m_{12}, m_{13})$ x10x
(m_4, m_5) 010x ✓		
(m_4, m_{12}) x100 ✓		
(m_5, m_{13}) x101 ✓	2	
(m_{12}, m_{13}) 110x ✓		
(m_{13}, m_{15}) 11x1	3	

For our example, notice that the third term in Group 1 and the second term in Group 2 meet this requirement, differing only with the *A* literal. The fourth term in Group 1 also can be combined with the first term in Group 2, forming a redundant set of minterms. One of these can be crossed off the list and will not be used in the final expression.

With complicated expressions, the process described can be continued. For our example, we can read the *Second Level* expression as $B\overline{C}$. The terms that are unchecked will form other terms in the final reduced expression. The first unchecked term is read as $\overline{A}\overline{B}D$. The next one is read as $\overline{A}\overline{C}D$. The last unchecked term is *ABD*. Recall that m_{10} was an essential prime implicant, so is picked up in the final expression. The reduced expression using the unchecked terms is:

$$X = B\overline{C} + \overline{A}\overline{B}D + \overline{A}\overline{C}D + ABD + A\overline{B}C\overline{D}$$

Although this expression is correct, it may not be the minimum possible expression. There is a final check that can eliminate any unnecessary terms. The terms for the expression are written into a prime implicant table, with minterms for each prime implicant checked, as shown in Table 4–13.

▶ TABLE 4-13

	Minterms							
Prime Implicants	m_1	m_3	m_4	m_5	m_{10}	m_{12}	m_{13}	m_{15}
$B\overline{C}$ $(m_4, m_5, m_{12}, m_{13})$			✓	✓		✓	✓	
$\overline{A}\overline{B}D$ (m_1, m_3)	✓	✓						
$\overline{A}\overline{C}D$ (m_1, m_5)	✓			✓				
ABD (m_{13}, m_{15})							✓	✓
$A\overline{B}C\overline{D}$ (m_{10})					✓			

If a minterm has a single check mark, then the prime implicant is essential and must be included in the final expression. The term *ABD* must be included because m_{15} is only covered by it. Likewise m_{10} is only covered by $A\overline{B}C\overline{D}$, so it must be in the final expression. Notice that the two minterms in $\overline{A}\overline{C}D$ are covered by the prime implicants in the first two rows, so this term is unnecessary. The final reduced expression is, therefore,

$$X = B\overline{C} + \overline{A}\overline{B}D + ABD + A\overline{B}C\overline{D}$$

SECTION 4-11 CHECKUP

1. What is a minterm?
2. What is an essential prime implicant?

TRUE/FALSE QUIZ

Answers are at the end of the chapter.

1. *Variable, complement,* and *literal* are all terms used in Boolean algebra.
2. Addition in Boolean algebra is equivalent to the OR function.
3. Multiplication in Boolean algebra is equivalent to the NAND function.
4. The commutative law, associative law, and distributive law are all laws in Boolean algebra.
5. The complement of a 1 is 0.
6. When a Boolean variable is multiplied by its complement, the result is the variable.

7. "The complement of a product of variables is equal to the sum of the complements of each variable" is a statement of DeMorgan's theorem.

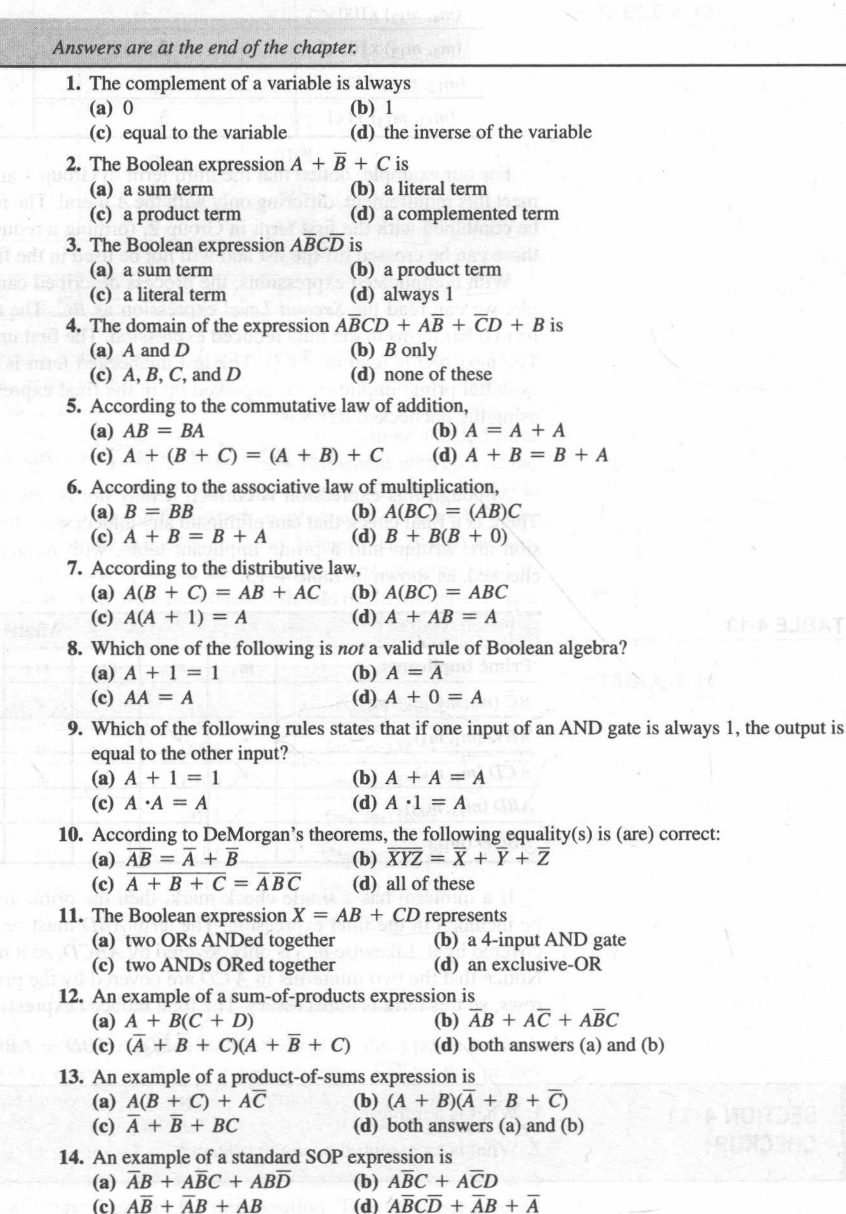
8. SOP means series of products.

9. Karnaugh maps can be used to simplify Boolean expressions.

10. A 4-variable Karnaugh map has eight cells.

SELF-TEST

Answers are at the end of the chapter.



1. The complement of a variable is always
- **(a)** 0 **(b)** 1
- **(c)** equal to the variable **(d)** the inverse of the variable

2. The Boolean expression $A + \overline{B} + C$ is
- **(a)** a sum term **(b)** a literal term
- **(c)** a product term **(d)** a complemented term

3. The Boolean expression $A\overline{B}C\overline{D}$ is
- **(a)** a sum term **(b)** a product term
- **(c)** a literal term **(d)** always 1

4. The domain of the expression $A\overline{B}CD + A\overline{B} + \overline{C}D + B$ is
- **(a)** A and D **(b)** B only
- **(c)** A, B, C, and D **(d)** none of these

5. According to the commutative law of addition,
- **(a)** $AB = BA$ **(b)** $A = A + A$
- **(c)** $A + (B + C) = (A + B) + C$ **(d)** $A + B = B + A$

6. According to the associative law of multiplication,
- **(a)** $B = BB$ **(b)** $A(BC) = (AB)C$
- **(c)** $A + B = B + A$ **(d)** $B + B(B + 0)$

7. According to the distributive law,
- **(a)** $A(B + C) = AB + AC$ **(b)** $A(BC) = ABC$
- **(c)** $A(A + 1) = A$ **(d)** $A + AB = A$

8. Which one of the following is *not* a valid rule of Boolean algebra?
- **(a)** $A + 1 = 1$ **(b)** $A = \overline{A}$
- **(c)** $AA = A$ **(d)** $A + 0 = A$

9. Which of the following rules states that if one input of an AND gate is always 1, the output is equal to the other input?
- **(a)** $A + 1 = 1$ **(b)** $A + A = A$
- **(c)** $A \cdot A = A$ **(d)** $A \cdot 1 = A$

10. According to DeMorgan's theorems, the following equality(s) is (are) correct:
- **(a)** $\overline{AB} = \overline{A} + \overline{B}$ **(b)** $\overline{XYZ} = \overline{X} + \overline{Y} + \overline{Z}$
- **(c)** $\overline{A + B + C} = \overline{A}\,\overline{B}\,\overline{C}$ **(d)** all of these

11. The Boolean expression $X = AB + CD$ represents
- **(a)** two ORs ANDed together **(b)** a 4-input AND gate
- **(c)** two ANDs ORed together **(d)** an exclusive-OR

12. An example of a sum-of-products expression is
- **(a)** $A + B(C + D)$ **(b)** $\overline{A}B + A\overline{C} + A\overline{B}C$
- **(c)** $(\overline{A} + B + C)(A + \overline{B} + C)$ **(d)** both answers (a) and (b)

13. An example of a product-of-sums expression is
- **(a)** $A(B + C) + A\overline{C}$ **(b)** $(A + B)(\overline{A} + B + \overline{C})$
- **(c)** $\overline{A} + \overline{B} + BC$ **(d)** both answers (a) and (b)

14. An example of a standard SOP expression is
- **(a)** $\overline{A}B + A\overline{B}C + AB\overline{D}$ **(b)** $A\overline{B}C + A\overline{C}D$
- **(c)** $A\overline{B} + \overline{A}B + AB$ **(d)** $A\overline{B}C\overline{D} + \overline{A}B + \overline{A}$

15. A 3-variable Karnaugh map has
- **(a)** eight cells **(b)** three cells
- **(c)** sixteen cells **(d)** four cells

16. In a 4-variable Karnaugh map, a 2-variable product term is produced by
- **(a)** a 2-cell group of 1s **(b)** an 8-cell group of 1s
- **(c)** a 4-cell group of 1s **(d)** a 4-cell group of 0s

17. The Quine-McCluskey method can be used to
- **(a)** replace the Karnaugh map method **(b)** simplify expressions with 5 or more variables
- **(c)** both (a) and (b) **(d)** none of the above

PROBLEMS

Answers to odd-numbered problems are at the end of the book.

Section 4–1 Boolean Operations and Expressions

1. Using Boolean notation, write an expression that is a 1 whenever one or more of its variables (A, B, C, and D) are 1s.
2. Write an expression that is a 1 only if all of its variables (A, B, C, D, and E) are 1s.
3. Write an expression that is a 1 when one or more of its variables (A, B, and C) are 0s.
4. Evaluate the following operations:
   (a) $0 + 0 + 1$ **(b)** $1 + 1 + 1$ **(c)** $1 \cdot 0 \cdot 0$
   (d) $1 \cdot 1 \cdot 1$ **(e)** $1 \cdot 0 \cdot 1$ **(f)** $1 \cdot 1 + 0 \cdot 1 \cdot 1$
5. Find the values of the variables that make each product term 1 and each sum term 0.
   (a) AB **(b)** $A\overline{B}C$ **(c)** $A + B$ **(d)** $\overline{A} + B + \overline{C}$
   (e) $\overline{A} + \overline{B} + C$ **(f)** $\overline{A} + B$ **(g)** $A\overline{B}\,\overline{C}$
6. Find the value of X for all possible values of the variables.
   (a) $X = (A + B)C + B$ **(b)** $X = \overline{(A + B)}C$ **(c)** $X = A\overline{B}C + AB$
   (d) $X = (A + B)(\overline{A} + B)$ **(e)** $X = (A + B\overline{C})(\overline{B} + \overline{C})$

Section 4–2 Laws and Rules of Boolean Algebra

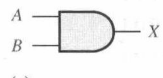
7. Identify the law of Boolean algebra upon which each of the following equalities is based:
   (a) $A\overline{B} + CD + A\overline{C}D + B = B + A\overline{B} + A\overline{C}D + CD$
   (b) $AB\overline{C}D + \overline{ABC} = D\overline{C}BA + \overline{CBA}$
   (c) $AB(CD + E\overline{F} + GH) = ABCD + ABE\overline{F} + ABGH$
8. Identify the Boolean rule(s) on which each of the following equalities is based:
   (a) $\overline{\overline{AB + CD}} + \overline{EF} = AB + CD + \overline{EF}$ **(b)** $A\overline{A}B + AB\overline{C} + AB\overline{B} = AB\overline{C}$
   (c) $A(BC + BC) + AC = A(BC) + AC$ **(d)** $AB(C + \overline{C}) + AC = AB + AC$
   (e) $A\overline{B} + A\overline{B}C = A\overline{B}$ **(f)** $ABC + \overline{AB} + \overline{ABC}D = ABC + \overline{AB} + D$

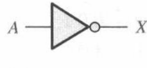
Section 4–3 DeMorgan's Theorems

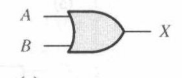
9. Apply DeMorgan's theorems to each expression:
   (a) $\overline{A + \overline{B}}$ **(b)** $\overline{\overline{A}B}$ **(c)** $\overline{A + B + C}$ **(d)** $\overline{ABC}$
   (e) $\overline{A(B + C)}$ **(f)** $\overline{AB} + \overline{CD}$ **(g)** $\overline{AB + CD}$ **(h)** $\overline{(A + \overline{B})(\overline{C} + D)}$
10. Apply DeMorgan's theorems to each expression:
    (a) $\overline{A\overline{B}(C + \overline{D})}$ **(b)** $\overline{AB(CD + EF)}$
    (c) $\overline{(A + \overline{B} + C + \overline{D})} + \overline{ABC\overline{D}}$ **(d)** $\overline{\overline{(\overline{A} + B + C + D)}\,\overline{(A\overline{B}\,\overline{C}D)}}$
    (e) $\overline{\overline{AB}(CD + \overline{E}F)(\overline{AB} + \overline{CD})}$
11. Apply DeMorgan's theorems to the following:
    (a) $\overline{\overline{(\overline{ABC})(\overline{EFG})} + \overline{(\overline{HIJ})(\overline{KLM})}}$ **(b)** $\overline{(A + \overline{B\overline{C}} + CD)} + \overline{\overline{BC}}$
    (c) $\overline{\overline{\overline{(A + B)}\,\overline{(C + D)}\,\overline{(E + F)}\,\overline{(G + H)}}}$

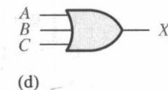
Section 4–4 Boolean Analysis of Logic Circuits

12. Write the Boolean expression for each of the logic gates in Figure 4–48.



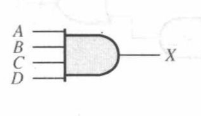


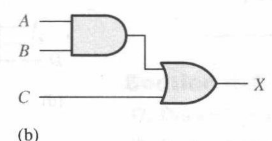


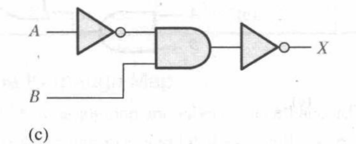


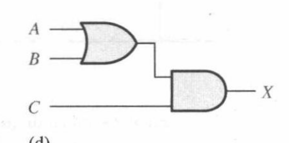
▶ **FIGURE 4-48**

13. Write the Boolean expression for each of the logic circuits in Figure 4–49.









▲ **FIGURE 4-49**

14. Draw the logic circuit represented by each of the following expressions:

(a) $A + B + C$ **(b)** ABC
(c) $AB + C$ **(d)** $AB + CD$

15. Draw the logic circuit represented by each expression:

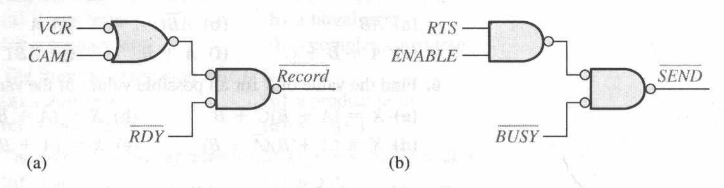
(a) $A\overline{B} + \overline{A}B$ **(b)** $AB + \overline{A}\,\overline{B} + \overline{A}BC$
(c) $\overline{A}B(C + \overline{D})$ **(d)** $A + B[C + D(B + \overline{C})]$

16. (a) Draw a logic circuit for the case where the output, ENABLE, is LOW only if the inputs, ASSERT and READY, are both HIGH.

(b) Draw a logic circuit for the case where the output, HOLD, is LOW only if the input, LOAD, is HIGH and the input, READY, is LOW.

17. Develop the truth table for each of the circuits in Figure 4–50.

▶ **FIGURE 4-50**



18. Construct a truth table for each of the following Boolean expressions:

(a) $A + B$ **(b)** AB **(c)** $AB + BC$
(d) $(A + B)C$ **(e)** $(A + B)(\overline{B} + C)$

Section 4–5 Logic Simplification Using Boolean Algebra

19. Using Boolean algebra techniques, simplify the following expressions as much as possible:

(a) $A(A + B)$ **(b)** $A(\overline{A} + AB)$ **(c)** $BC + \overline{B}C$
(d) $A(A + \overline{A}B)$ **(e)** $A\overline{B}C + \overline{A}BC + \overline{A}\,\overline{B}C$

20. Using Boolean algebra, simplify the following expressions:

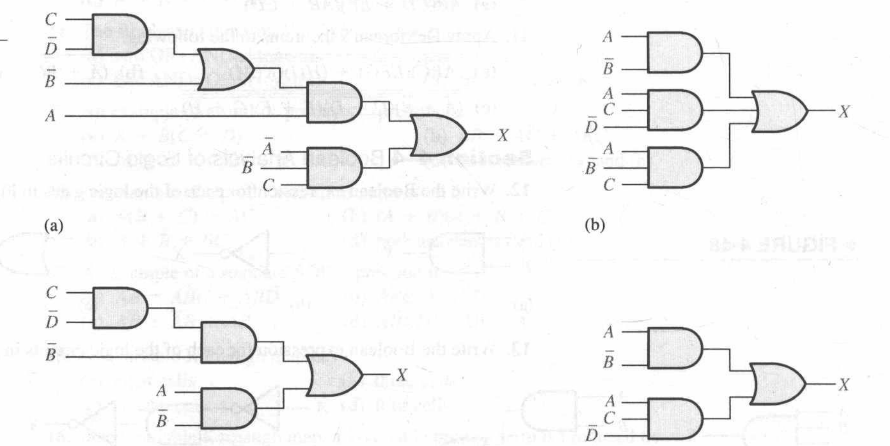
(a) $(A + \overline{B})(A + C)$ **(b)** $\overline{A}B + \overline{A}B\overline{C} + \overline{A}BCD + \overline{A}B\overline{C}\,\overline{D}E$
(c) $AB + \overline{AB}C + A$ **(d)** $(A + \overline{A})(AB + AB\overline{C})$
(e) $AB + (\overline{A} + \overline{B})C + AB$

21. Using Boolean algebra, simplify each expression:

(a) $BD + B(D + E) + \overline{D}(D + F)$ **(b)** $\overline{A}\,\overline{B}C + \overline{(A + B + \overline{C})} + \overline{A}\,\overline{B}\,\overline{C}D$
(c) $(B + BC)(B + \overline{B}C)(B + D)$ **(d)** $ABCD + AB(\overline{CD}) + (\overline{AB})CD$
(e) $ABC[AB + \overline{C}(BC + AC)]$

22. Determine which of the logic circuits in Figure 4–51 are equivalent.

▶ **FIGURE 4-51**



Section 4–6 Standard Forms of Boolean Expressions

23. Convert the following expressions to sum-of-product (SOP) forms:
(a) $(A + B)(C + \overline{B})$ **(b)** $(A + \overline{B}C)C$ **(c)** $(A + C)(AB + AC)$

24. Convert the following expressions to sum-of-product (SOP) forms:
(a) $AB + CD(A\overline{B} + CD)$ **(b)** $AB(\overline{B}\,\overline{C} + BD)$ **(c)** $A + B[AC + (B + \overline{C})D]$

25. Define the domain of each SOP expression in Problem 23 and convert the expression to standard SOP form.

26. Convert each SOP expression in Problem 24 to standard SOP form.

27. Determine the binary value of each term in the standard SOP expressions from Problem 25.

28. Determine the binary value of each term in the standard SOP expressions from Problem 26.

29. Convert each standard SOP expression in Problem 25 to standard POS form.

30. Convert each standard SOP expression in Problem 26 to standard POS form.

Section 4–7 Boolean Expressions and Truth Tables

31. Develop a truth table for each of the following standard SOP expressions:
(a) $A\overline{B}C + \overline{A}B\overline{C} + ABC$ **(b)** $\overline{X}\,\overline{Y}\,\overline{Z} + \overline{X}\,\overline{Y}Z + XY\overline{Z} + X\overline{Y}Z + \overline{X}YZ$

32. Develop a truth table for each of the following standard SOP expressions:
(a) $\overline{A}B\overline{C}D + \overline{A}BC\overline{D} + A\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$
(b) $WXYZ + WXY\overline{Z} + \overline{W}XYZ + W\overline{X}YZ + WX\overline{Y}Z$

33. Develop a truth table for each of the SOP expressions:
(a) $\overline{A}B + AB\overline{C} + \overline{A}\,\overline{C} + A\overline{B}C$ **(b)** $\overline{X} + Y\overline{Z} + WZ + X\overline{Y}Z$

34. Develop a truth table for each of the standard POS expressions:
(a) $(\overline{A} + \overline{B} + \overline{C})(A + B + C)(A + \overline{B} + C)$
(b) $(\overline{A} + B + \overline{C} + D)(A + \overline{B} + C + \overline{D})(A + \overline{B} + \overline{C} + D)(\overline{A} + B + C + \overline{D})$

35. Develop a truth table for each of the standard POS expressions:
(a) $(A + B)(A + C)(A + B + C)$
(b) $(A + \overline{B})(A + \overline{B} + \overline{C})(B + C + \overline{D})(\overline{A} + B + \overline{C} + D)$

36. For each truth table in Table 4–14, derive a standard SOP and a standard POS expression.

TABLE 4-14

ABC	*X*
000	0
001	1
010	0
011	0
100	1
101	1
110	0
111	1

(a)

ABC	*X*
000	0
001	0
010	0
011	0
100	0
101	1
110	1
111	1

(b)

ABCD	*X*
0000	1
0001	1
0010	0
0011	1
0100	0
0101	1
0110	1
0111	0
1000	0
1001	1
1010	0
1011	0
1100	1
1101	0
1110	0
1111	0

(c)

ABCD	*X*
0000	0
0001	0
0010	1
0011	0
0100	1
0101	1
0110	0
0111	1
1000	0
1001	0
1010	0
1011	1
1100	1
1101	0
1110	0
1111	1

(d)

Section 4–8 The Karnaugh Map

37. Draw a 3-variable Karnaugh map and label each cell according to its binary value.

38. Draw a 4-variable Karnaugh map and label each cell according to its binary value.

39. Write the standard product term for each cell in a 3-variable Karnaugh map.

Section 4–9 Karnaugh Map SOP Minimization

40. Use a Karnaugh map to find the minimum SOP form for each expression:

(a) $\bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + A\bar{B}C$ (b) $AC(\bar{B} + C)$

(c) $\bar{A}(BC + B\bar{C}) + A(BC + B\bar{C})$ (d) $\bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + \bar{A}B\bar{C} + AB\bar{C}$

41. Use a Karnaugh map to simplify each expression to a minimum SOP form:

(a) $\bar{A}\bar{B}\bar{C} + A\bar{B}C + \bar{A}BC + AB\bar{C}$ (b) $AC[\bar{B} + B(B + \bar{C})]$

(c) $DE\bar{F} + \bar{D}E\bar{F} + \bar{D}\bar{E}\bar{F}$

42. Expand each expression to a standard SOP form:

(a) $AB + A\bar{B}C + ABC$ (b) $A + BC$

(c) $A\bar{B}\bar{C}D + AC\bar{D} + B\bar{C}D + \bar{A}BC\bar{D}$ (d) $A\bar{B} + A\bar{B}\,\bar{C}D + CD + B\bar{C}D + ABCD$

43. Minimize each expression in Problem 42 with a Karnaugh map.

44. Use a Karnaugh map to reduce each expression to a minimum SOP form:

(a) $A + B\bar{C} + CD$

(b) $\bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D + ABCD + ABC\bar{D}$

(c) $\bar{A}B(\bar{C}\bar{D} + \bar{C}D) + AB(\bar{C}\bar{D} + \bar{C}D) + A\bar{B}\bar{C}D$

(d) $(\bar{A}\bar{B} + A\bar{B})(CD + C\bar{D})$

(e) $\bar{A}\bar{B} + A\bar{B} + \bar{C}\bar{D} + C\bar{D}$

45. Reduce the function specified in truth Table 4–15 to its minimum SOP form by using a Karnaugh map.

46. Use the Karnaugh map method to implement the minimum SOP expression for the logic function specified in truth Table 4–16.

47. Solve Problem 46 for a situation in which the last six binary combinations are not allowed.

TABLE 4-15

Inputs			Output
A	*B*	*C*	*X*
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

TABLE 4-16

Inputs				Output
A	*B*	*C*	*D*	*X*
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

Section 4–10 Karnaugh Map POS Minimization

48. Use a Karnaugh map to find the minimum POS for each expression:

(a) $(A + B + C)(\bar{A} + \bar{B} + \bar{C})(A + \bar{B} + C)$

(b) $(X + \bar{Y})(\bar{X} + Z)(X + \bar{Y} + \bar{Z})(\bar{X} + \bar{Y} + Z)$

(c) $A(B + \bar{C})(\bar{A} + C)(A + \bar{B} + C)(\bar{A} + B + \bar{C})$

49. Use a Karnaugh map to simplify each expression to minimum POS form:

(a) $(A + \bar{B} + C + \bar{D})(\bar{A} + B + \bar{C} + D)(\bar{A} + \bar{B} + \bar{C} + \bar{D})$

(b) $(X + \bar{Y})(W + \bar{Z})(\bar{X} + \bar{Y} + \bar{Z})(W + X + Y + Z)$

50. For the function specified in Table 4–15, determine the minimum POS expression using a Karnaugh map.

51. Determine the minimum POS expression for the function in Table 4–16.

52. Convert each of the following POS expressions to minimum SOP expressions using a Karnaugh map:

(a) $(A + \overline{B})(A + \overline{C})(\overline{A} + \overline{B} + C)$

(b) $(\overline{A} + B)(\overline{A} + \overline{B} + \overline{C})(B + \overline{C} + D)(A + \overline{B} + C + \overline{D})$

Section 4–11 The Quine-McCluskey Method

53. List the minterms in the expression

$$X = ABC + \overline{A}\overline{B}C + AB\overline{C} + A\overline{B}C + \overline{A}BC$$

54. List the minterms in the expression

$$X = \overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C}D + \overline{A}B\overline{C}D + AB\overline{C}\overline{D} + A\overline{B}C\overline{D} + \overline{A}BC\overline{D} + A\overline{B}\overline{C}D$$

55. Create a table for the number of 1s in the minterms for the expression in Problem 54 (similar to Table 4–10).

56. Create a table of first level minterms for the expression in Problem 54 (similar to Table 4–11).

57. Create a table of second level minterms for the expression in Problem 54 (similar to Table 4–12).

58. Create a table of prime implicants for the expression in Problem 54 (similar to Table 4–13).

59. Determine the final reduced expression for the expression in Problem 54.

ANSWERS

SECTION CHECKUPS

Section 4–1 Boolean Operations and Expressions

1. $\overline{A} = \overline{0} = 1$
2. $A = 1, B = 1, C = 0; \overline{A} + \overline{B} + C = \overline{1} + \overline{1} + 0 = 0 + 0 + 0 = 0$
3. $A = 1, B = 0, C = 1; A\overline{B}C = 1 \cdot \overline{0} \cdot 1 = 1 \cdot 1 \cdot 1 =$

Section 4–2 Laws and Rules of Boolean Algebra

1. $A + (B + C + D) = (A + B + C) + D$
2. $A(B + C + D) = AB + AC + AD$

Section 4–3 DeMorgan's Theorems

1. **(a)** $\overline{ABC} + \overline{(\overline{D} + E)} = \overline{A} + \overline{B} + \overline{C} + D\overline{E}$ **(b)** $\overline{(A + B)C} = \overline{A}\,\overline{B} + \overline{C}$

   (c) $\overline{A + B + C} + \overline{\overline{D}E} = \overline{A}\,\overline{B}\,\overline{C} + D + \overline{E}$

Section 4–4 Boolean Analysis of Logic Circuits

1. $(C + D)B + A$
2. Abbreviated truth table: The expression is a 1 when A is 1 or when B and C are 1s or when B and D are 1s. The expression is 0 for all other variable combinations.

Section 4–5 Logic Simplification Using Boolean Algebra

1. **(a)** $A + AB + A\overline{B}C = A$ **(b)** $(\overline{A} + B)C + ABC = C(\overline{A} + B)$

   (c) $A\overline{B}C(BD + CDE) + A\overline{C} = A(\overline{C} + \overline{B}DE)$
2. **(a)** *Original:* 2 AND gates, 1 OR gate, 1 inverter; *Simplified:* No gates (straight connection)

   (b) *Original:* 2 OR gates, 2 AND gates, 1 inverter; *Simplified:* 1 OR gate, 1 AND gate, 1 inverter

   (c) *Original:* 5 AND gates, 2 OR gates, 2 inverters; *Simplified:* 2 AND gates, 1 OR gate, 2 inverters

Section 4–6 Standard Forms of Boolean Expressions

1. **(a)** SOP **(b)** standard POS **(c)** standard SOP **(d)** POS
2. **(a)** $AB\overline{C}\overline{D} + AB\overline{C}D + ABC\overline{D} + ABCD + \overline{A}B\overline{C}D + \overline{A}BCD + \overline{A}\,\overline{B}C\overline{D} + \overline{A}BC\overline{D}$

   (c) Already standard
3. **(b)** Already standard

   (d) $(A + \overline{B} + \overline{C})(A + \overline{B} + C)(A + B + \overline{C})(A + B + C)$

Section 4–7 Boolean Expressions and Truth Tables

1. $2^5 = 32$ **2.** $0110 \longrightarrow \overline{W}XY\overline{Z}$ **3.** $1100 \longrightarrow \overline{W} + \overline{X} + Y + Z$

Section 4–8 The Karnaugh Map

1. (a) upper left cell: 000 **(b)** lower right cell: 101
(c) lower left cell: 100 **(d)** upper right cell: 001

2. (a) upper left cell: $\overline{X}\overline{Y}\overline{Z}$ **(b)** lower right cell: $X\overline{Y}Z$
(c) lower left cell: $X\overline{Y}\overline{Z}$ **(d)** upper right cell: $\overline{X}\overline{Y}Z$

3. (a) upper left cell: 0000 **(b)** lower right cell: 1010
(c) lower left cell: 1000 **(d)** upper right cell: 0010

4. (a) upper left cell: $\overline{W}\overline{X}\overline{Y}\overline{Z}$ **(b)** lower right cell: $W\overline{X}Y\overline{Z}$
(c) lower left cell: $W\overline{X}\overline{Y}\overline{Z}$ **(d)** upper right cell: $\overline{W}\overline{X}Y\overline{Z}$

Section 4–9 Karnaugh Map SOP Minimization

1. 8-cell map for 3 variables; 16-cell map for 4 variables

2. $AB + B\overline{C} + \overline{A}\overline{B}C$

3. (a) $\overline{A}\overline{B}\overline{C} + \overline{A}BC + ABC + AB\overline{C}$

(b) $\overline{A}\overline{B}\overline{C} + \overline{A}\overline{B}C + \overline{A}B\overline{C} + ABC + AB\overline{C} + A\overline{B}C$

(c) $\overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C}D + \overline{A}B\overline{C}\overline{D} + \overline{A}B\overline{C}D + \overline{A}BCD + \overline{A}BC\overline{D} + A\overline{B}\overline{C}D + A\overline{B}CD$

(d) $\overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}B\overline{C}\overline{D} + AB\overline{C}\overline{D} + A\overline{B}\overline{C}\overline{D} + \overline{A}B\overline{C}D + AB\overline{C}D + A\overline{B}CD + \overline{A}\overline{B}C\overline{D} +$
$\overline{A}BC\overline{D} + ABC\overline{D} + A\overline{B}C\overline{D}$

Section 4–10 Karnaugh Map POS Minimization

1. In mapping a POS expression, 0s are placed in cells whose value makes the standard sum term zero; and in mapping an SOP expression 1s are placed in cells having the same values as the product terms.

2. 0 in the 1011 cell: $\overline{A} + B + \overline{C} + \overline{D}$

3. 1 in the 0010 cell: $\overline{A}\overline{B}C\overline{D}$

Section 4–11 The Quine-McCluskey Method

1. A minterm is a product term in which each variable appears once, either complemented or uncomplemented.

2. An essential prime implicant is a product term that cannot be further simplified by combining with other terms.

RELATED PROBLEMS FOR EXAMPLES

4–1 $\overline{A} + B = 0$ when $A = 1$ and $B = 0$.

4–2 $\overline{A}\overline{B} = 1$ when $A = 0$ and $B = 0$.

4–3 XYZ

4–4 $W + X + Y + Z$

4–5 $ABC\overline{D}\overline{E}$

4–6 $(A + \overline{B} + \overline{C}D)\overline{E}$

4–7 $\overline{ABCD} = \overline{A} + \overline{B} + \overline{C} + \overline{D}$

4–8 Results should be same as example.

4–9 $A\overline{B}$

4–10 CD

4–11 $AB\overline{C} + \overline{A}C + \overline{A}\overline{B}$

4–12 $\overline{A} + \overline{B} + \overline{C}$

4–13 Results should be same as example.

4–14 $\overline{A}B\overline{C} + AB + A\overline{C} + A\overline{B} + \overline{B}\overline{C}$

4–15 $W\overline{X}YZ + W\overline{X}Y\overline{Z} + W\overline{X}Y\overline{Z} + \overline{W}\overline{X}Y\overline{Z} + WX\overline{Y}Z + WX\overline{Y}\overline{Z}$

4–16 011, 101, 110, 010, 111. Yes

4–17 $(A + \overline{B} + C)(A + \overline{B} + \overline{C})(A + B + C)(\overline{A} + B + C)$

4–18 010, 100, 001, 111, 011. Yes

4–19 SOP and POS expressions are equivalent.

4–20 See Table 4–17.

4–21 See Table 4–18.

▼TABLE 4-17

A	*B*	*C*	*X*
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

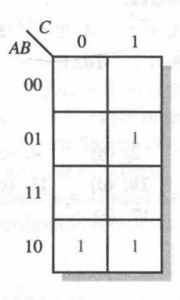
▼TABLE 4-18

A	*B*	*C*	*X*
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

4–22 The SOP and POS expressions are equivalent.

4–23 See Figure 4–52.

4–24 See Figure 4–53.



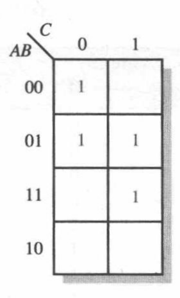
▲FIGURE 4-52

AB \ *CD*	00	01	11	10
00				
01				1
11	1		1	1
10				

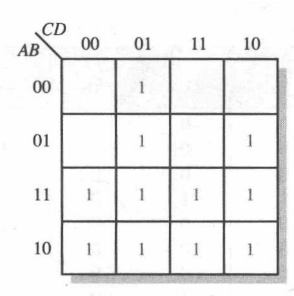
▲FIGURE 4-53

4–25 See Figure 4–54.

4–26 See Figure 4–55.



▲FIGURE 4-54



▲FIGURE 4-55

4–27 No other ways

4–28 $X = B + \overline{A}C + A\overline{C}D + C\overline{D}$

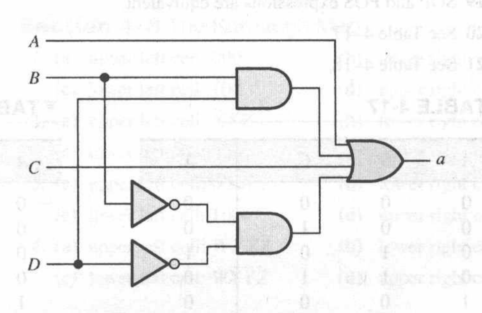
4–29 $X = \overline{D} + A\overline{B}C + B\overline{C} + \overline{A}B$

4–30 $Q = X + Y$

4–31 $Q = \overline{X}\overline{Y}\overline{Z} + W\overline{X}Z + \overline{W}YZ$

4–32 See Figure 4–56.

4–33 See Figure 4–57.



▲ **FIGURE 4-56**

AB \ CD	00	01	11	10
00	0	0		
01				0
11				
10				0

▲ **FIGURE 4-57**

4–34 $(X + \overline{Y})(X + \overline{Z})(\overline{X} + Y + Z)$

4–35 $(\overline{X} + \overline{Y} + Z)(\overline{W} + \overline{X} + Z)(W + X + Y + Z)(W + \overline{X} + Y + \overline{Z})$

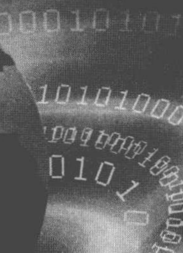
4–36 $\overline{Y}\overline{Z} + \overline{X}\overline{Z} + \overline{W}Y + \overline{X}\overline{Y}Z$

TRUE/FALSE QUIZ

1. T **2.** T **3.** F **4.** T **5.** T **6.** F
7. T **8.** F **9.** T **10.** F

SELF-TEST

1. (d) **2.** (a) **3.** (b) **4.** (c) **5.** (d) **6.** (b) **7.** (a)
8. (b) **9.** (d) **10.** (d) **11.** (c) **12.** (b) **13.** (b) **14.** (c)
15. (a) **16.** (c) **17.** (c)



Chapter 5 Combinational Logic Analysis

CHAPTER OUTLINE

5-1 Basic Combinational Logic Circuits

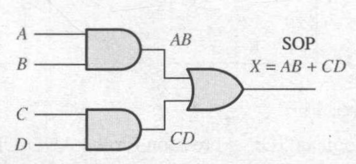
AND-OR Logic

AND-OR logic produces an SOP expression.

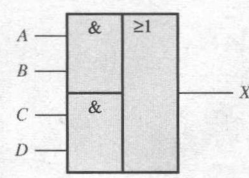
Figure 5–1(a) shows an AND-OR circuit consisting of two 2-input AND gates and one 2-input OR gate; Figure 5–1(b) is the ANSI standard rectangular outline symbol. The Boolean expressions for the AND gate outputs and the resulting SOP expression for the output X are shown on the diagram. In general, an AND-OR circuit can have any number of AND gates, each with any number of inputs.

The truth table for a 4-input AND-OR logic circuit is shown in Table 5–1. The intermediate AND gate outputs (the AB and CD columns) are also shown in the table.

MultiSim



(a) Logic diagram (ANSI standard distinctive shape symbols)



(b) ANSI standard rectangular outline symbol

▶ **FIGURE 5-1**
An example of AND-OR logic. Open file F05-01 to verify the operation. *A multisim tutorial is available on the website.*

▶ **TABLE 5-1**
Truth table for the AND-OR logic in Figure 5–1.

Inputs						Output
A	*B*	*C*	*D*	*AB*	*CD*	*X*
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	1	1
0	1	0	0	0	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	1	1	0	1	1
1	0	0	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	0	0	0
1	0	1	1	0	1	1
1	1	0	0	1	0	1
1	1	0	1	1	0	1
1	1	1	0	1	0	1
1	1	1	1	1	1	1

An AND-OR circuit directly implements an SOP expression, assuming the complements (if any) of the variables are available. The operation of the AND-OR circuit in Figure 5–1 is stated as follows:

For a 4-input AND-OR logic circuit, the output *X* is HIGH (1) if both input *A* and input *B* are HIGH (1) or both input *C* and input *D* are HIGH (1).

EXAMPLE 5-1

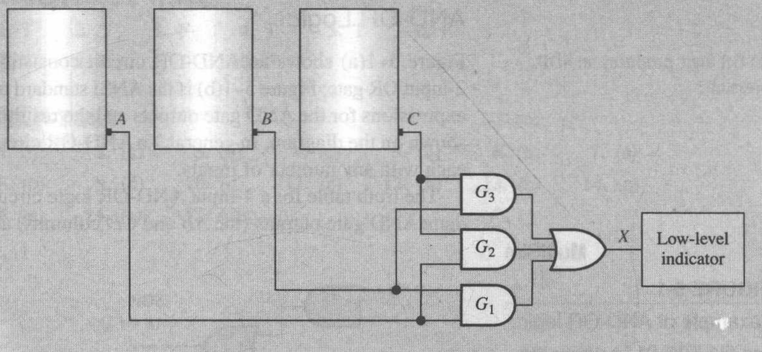
In a certain chemical-processing plant, a liquid chemical is used in a manufacturing process. The chemical is stored in three different tanks. A level sensor in each tank produces a HIGH voltage when the level of chemical in the tank drops below a specified point.

Design a circuit that monitors the chemical level in each tank and indicates when the level in any two of the tanks drops below the specified point.

Solution

The AND-OR circuit in Figure 5–2 has inputs from the sensors on tanks *A*, *B*, and *C* as shown. The AND gate G_1 checks the levels in tanks *A* and *B*, gate G_2 checks tanks *A* and *C*, and gate G_3 checks tanks *B* and *C*. When the chemical level in any two of the tanks gets too low, one of the AND gates will have HIGHs on both of its inputs, causing its output to be HIGH; and so the final output *X* from the OR gate is HIGH. This HIGH input is then used to activate an indicator such as a lamp or audible alarm, as shown in the figure.

▶ FIGURE 5-2



Related Problem*

Write the Boolean SOP expression for the AND-OR logic in Figure 5–2.

*Answers are at the end of the chapter.

AND-OR-Invert Logic

When the output of an AND-OR circuit is complemented (inverted), it results in an AND-OR-Invert circuit. Recall that AND-OR logic directly implements SOP expressions. POS expressions can be implemented with AND-OR-Invert logic. This is illustrated as follows, starting with a POS expression and developing the corresponding AND-OR-Invert (AOI) expression.

$$X = (\overline{A} + \overline{B})(\overline{C} + \overline{D}) = (\overline{AB})(\overline{CD}) = \overline{\overline{(\overline{AB})(\overline{CD})}} = \overline{\overline{\overline{AB}} + \overline{\overline{CD}}} = \overline{AB + CD}$$

The logic diagram in Figure 5–3(a) shows an AND-OR-Invert circuit with four inputs and the development of the POS output expression. The ANSI standard rectangular outline symbol is shown in part (b). In general, an AND-OR-Invert circuit can have any number of AND gates, each with any number of inputs.

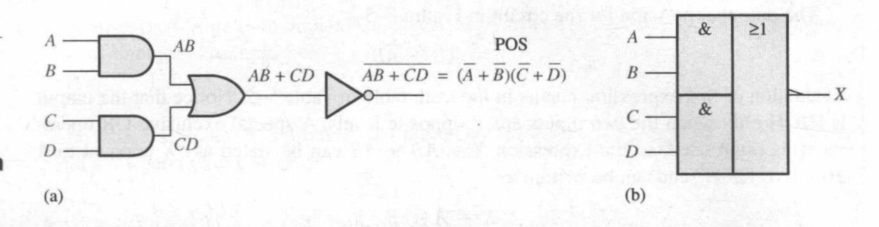
The operation of the AND-OR-Invert circuit in Figure 5–3 is stated as follows:

For a 4-input AND-OR-Invert logic circuit, the output *X* is LOW (0) if both input *A* and input *B* are HIGH (1) or both input *C* and input *D* are HIGH (1).

A truth table can be developed from the AND-OR truth table in Table 5–1 by simply changing all 1s to 0s and all 0s to 1s in the output column.

▶ **FIGURE 5-3**

An AND-OR-Invert circuit produces a POS output. Open file F05-03 to verify the operation. **MultiSim**



EXAMPLE 5-2

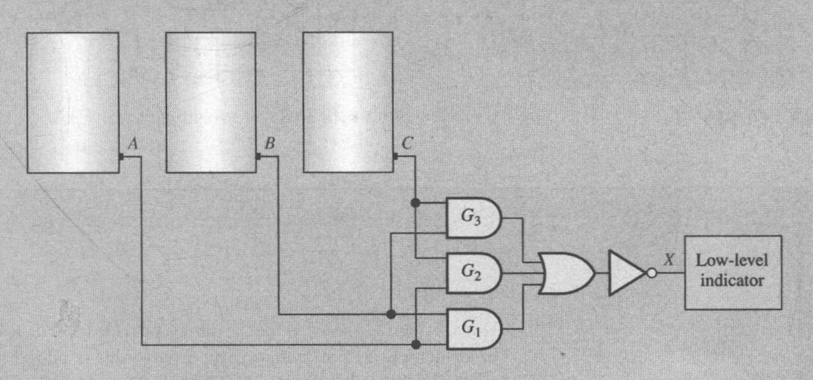
The sensors in the chemical tanks of Example 5–1 are being replaced by a new model that produces a LOW voltage instead of a HIGH voltage when the level of the chemical in the tank drops below a critical point.

Modify the circuit in Figure 5–2 to operate with the different input levels and still produce a HIGH output to activate the indicator when the level in any two of the tanks drops below the critical point. Show the logic diagram.

Solution

The AND-OR-Invert circuit in Figure 5–4 has inputs from the sensors on tanks *A*, *B*, and *C* as shown. The AND gate G_1 checks the levels in tanks *A* and *B*, gate G_2 checks tanks *A* and *C*, and gate G_3 checks tanks *B* and *C*. When the chemical level in any two of the tanks gets too low, each AND gate will have a LOW on at least one input, causing its output to be LOW and, thus, the final output *X* from the inverter is HIGH. This HIGH output is then used to activate an indicator.

▶ **FIGURE 5-4**



Related Problem

Write the Boolean expression for the AND-OR-Invert logic in Figure 5–4 and show that the output is HIGH (1) when any two of the inputs *A*, *B*, and *C* are LOW (0).

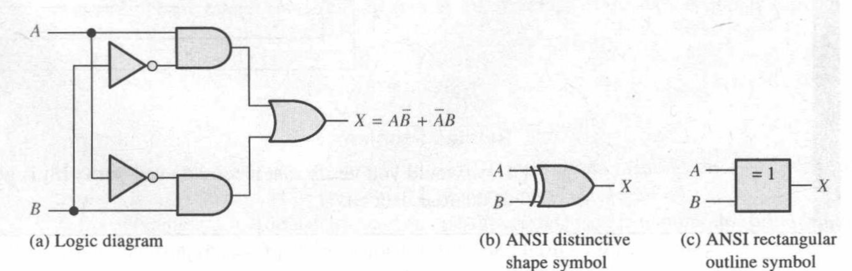
Exclusive-OR Logic

The XOR gate is actually a combination of other gates.

The exclusive-OR gate was introduced in Chapter 3. Although this circuit is considered a type of logic gate with its own unique symbol, it is actually a combination of two AND gates, one OR gate, and two inverters, as shown in Figure 5–5(a). The two ANSI standard exclusive-OR logic symbols are shown in parts (b) and (c).

▶ **FIGURE 5-5**

Exclusive-OR logic diagram and symbols. Open file F05-05 to verify the operation. **MultiSim**



(a) Logic diagram (b) ANSI distinctive shape symbol (c) ANSI rectangular outline symbol

The output expression for the circuit in Figure 5–5 is

$$X = A\overline{B} + \overline{A}B$$

Evaluation of this expression results in the truth table in Table 5–2. Notice that the output is HIGH only when the two inputs are at opposite levels. A special exclusive-OR operator $\oplus$ is often used, so the expression $X = A\overline{B} + \overline{A}B$ can be stated as "X is equal to A exclusive-OR B" and can be written as

$$X = A \oplus B$$

TABLE 5-2
Truth table for an exclusive-OR.

A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

Exclusive-NOR Logic

As you know, the complement of the exclusive-OR function is the exclusive-NOR, which is derived as follows:

$$X = \overline{A\overline{B} + \overline{A}B} = \overline{(A\overline{B})}\,\overline{(\overline{A}B)} = (\overline{A} + B)(A + \overline{B}) = \overline{A}\,\overline{B} + AB$$

Notice that the output X is HIGH only when the two inputs, A and B, are at the same level.

The exclusive-NOR can be implemented by simply inverting the output of an exclusive-OR, as shown in Figure 5–6(a), or by directly implementing the expression $\overline{A}\,\overline{B} + AB$, as shown in part (b).

(a) $X = \overline{A\overline{B} + \overline{A}B}$

(b) $X = \overline{A}\,\overline{B} + AB$

FIGURE 5-6
Two equivalent ways of implementing the exclusive-NOR. Open files F05-06 (a) and (b) to verify the operation.

MultiSim

EXAMPLE 5-3

Use exclusive-OR gates to implement an even-parity code generator for an original 4-bit code.

Solution

Recall from Chapter 2 that a parity bit is added to a binary code in order to provide error detection. For even parity, a parity bit is added to the original code to make the total number of 1s in the code even. The circuit in Figure 5–7 produces a 1 output when there is an odd number of 1s on the inputs in order to make the total number of 1s in the output code even. A 0 output is produced when there is an even number of 1s on the inputs.

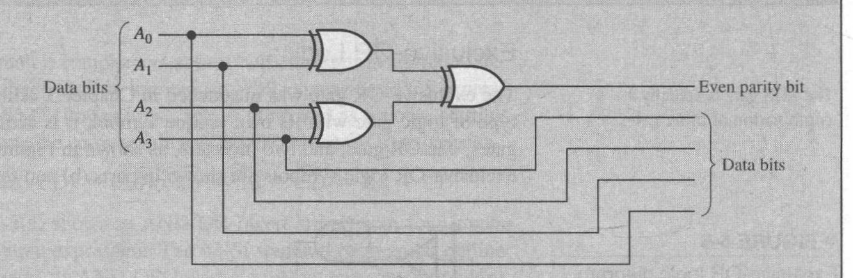


FIGURE 5-7
Even-parity generator.

Related Problem

How would you verify that a correct even-parity bit is generated for each combination of the four data bits?

EXAMPLE 5-4

Use exclusive-OR gates to implement an even-parity checker for the 5-bit code generated by the circuit in Example 5–3.

Solution

The circuit in Figure 5–8 produces a 1 output when there is an error in the five-bit code and a 0 when there is no error.

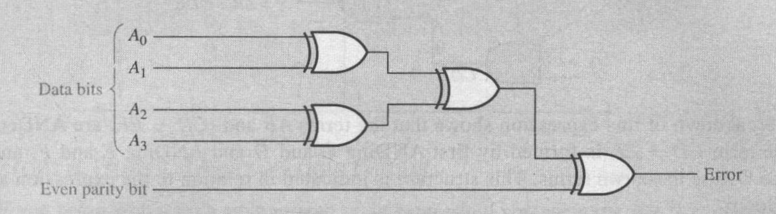


FIGURE 5-8
Even-parity checker.

Related Problem

How would you verify that an error is indicated when the input code is incorrect?

SECTION 5-1 CHECKUP

Answers are at the end of the chapter.

1. Determine the output (1 or 0) of a 4-variable AND-OR-Invert circuit for each of the following input conditions:
   (a) $A = 1, B = 0, C = 1, D = 0$ (b) $A = 1, B = 1, C = 0, D = 1$
   (c) $A = 0, B = 1, C = 1, D = 1$
2. Determine the output (1 or 0) of an exclusive-OR gate for each of the following input conditions:
   (a) $A = 1, B = 0$ (b) $A = 1, B = 1$
   (c) $A = 0, B = 1$ (d) $A = 0, B = 0$
3. Develop the truth table for a certain 3-input logic circuit with the output expression $X = A\overline{B}C + \overline{A}BC + \overline{A}\,\overline{B}\,\overline{C} + AB\overline{C} + ABC$.
4. Draw the logic diagram for an exclusive-NOR circuit.

5-2 Implementing Combinational Logic

For every Boolean expression there is a logic circuit, and for every logic circuit there is a Boolean expression.

InfoNote

Many control programs require logic operations to be performed by a computer. A driver program is a control program that is used with computer peripherals. For example, a mouse driver requires logic tests to determine if a button has been pressed and further logic operations to determine if it has moved, either horizontally or vertically. Within the heart of a microprocessor is the arithmetic logic unit (ALU), which performs

From a Boolean Expression to a Logic Circuit

Let's examine the following Boolean expression:

$$X = AB + CDE$$

A brief inspection shows that this expression is composed of two terms, AB and CDE, with a domain of five variables. The first term is formed by ANDing A with B, and the second term is formed by ANDing C, D, and E. The two terms are then ORed to form the output X. These operations are indicated in the structure of the expression as follows:

$$X = AB + CDE$$

(AND arrows point to AB and CDE; OR arrow points to $+$)

Note that in this particular expression, the AND operations forming the two individual terms, AB and CDE, must be performed *before* the terms can be ORed.

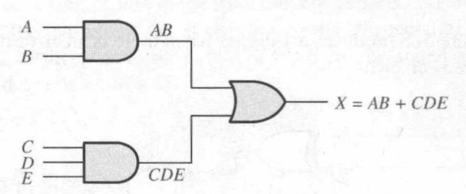
To implement this Boolean expression, a 2-input AND gate is required to form the term AB, and a 3-input AND gate is needed to form the term CDE. A 2-input OR gate is then

required to combine the two AND terms. The resulting logic circuit is shown in Figure 5–9.

As another example, let's implement the following expression:

$$X = AB(C\overline{D} + EF)$$

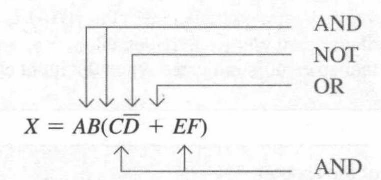
these logic operations as directed by program instructions. All of the logic described in this chapter can also be performed by the ALU, given the proper instructions.



◀ **FIGURE 5-9**

Logic circuit for $X = AB + CDE$.

A breakdown of this expression shows that the terms AB and $(C\overline{D} + EF)$ are ANDed. The term $C\overline{D} + EF$ is formed by first ANDing C and $\overline{D}$ and ANDing E and F, and then ORing these two terms. This structure is indicated in relation to the expression as follows:



Before you can implement the final expression, you must create the sum term $C\overline{D} + EF$ but before you can get this term; you must create the product terms $C\overline{D}$ and EF; but before you can get the term $C\overline{D}$, you must create $\overline{D}$. So, as you can see, the logic operations must be done in the proper order.

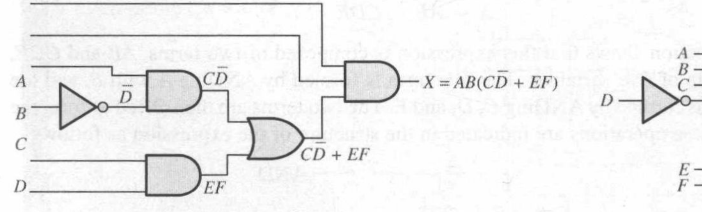
The logic gates required to implement $X = AB(C\overline{D} + EF)$ are as follows:

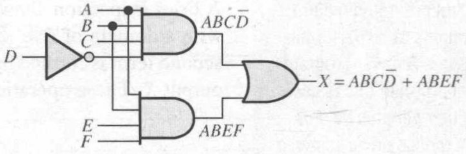
1. One inverter to form $\overline{D}$
2. Two 2-input AND gates to form $C\overline{D}$ and EF
3. One 2-input OR gate to form $C\overline{D} + EF$
4. One 3-input AND gate to form X

The logic circuit for this expression is shown in Figure 5–10(a). Notice that there is a maximum of four gates and an inverter between an input and output in this circuit (from input D to output). Often the total propagation delay time through a logic circuit is a major consideration. Propagation delays are additive, so the more gates or inverters between input and output, the greater the propagation delay time.

Unless an intermediate term, such as $C\overline{D} + EF$ in Figure 5–10(a), is required as an output for some other purpose, it is usually best to reduce a circuit to its SOP form in order to reduce the overall propagation delay time. The expression is converted to SOP as follows, and the resulting circuit is shown in Figure 5–10(b).

$$AB(C\overline{D} + EF) = ABC\overline{D} + ABEF$$





(b) Sum-of-products implementation of the circuit in part (a)

▲ **FIGURE 5-10**

Logic circuits for $X = AB(C\overline{D} + EF) = ABC\overline{D} + ABEF$.

▶ TABLE 5-3

Inputs			Output	
A	B	C	X	Product Term
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\overline{A}BC$
1	0	0	1	$A\overline{B}\,\overline{C}$
1	0	1	0	
1	1	0	0	
1	1	1	0	

From a Truth Table to a Logic Circuit

If you begin with a truth table instead of an expression, you can write the SOP expression from the truth table and then implement the logic circuit. Table 5–3 specifies a logic function.

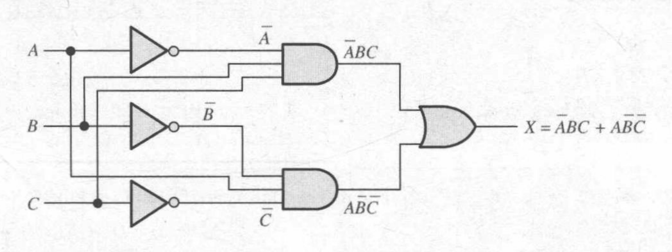
The Boolean SOP expression obtained from the truth table by ORing the product terms for which $X = 1$ is

$$X = \overline{A}BC + A\overline{B}\,\overline{C}$$

The first term in the expression is formed by ANDing the three variables $\overline{A}$, B, and C. The second term is formed by ANDing the three variables A, $\overline{B}$, and $\overline{C}$.

The logic gates required to implement this expression are as follows: three inverters to form the $\overline{A}$, $\overline{B}$, and $\overline{C}$ variables; two 3-input AND gates to form the terms $\overline{A}BC$ and $A\overline{B}\,\overline{C}$; and one 2-input OR gate to form the final output function, $\overline{A}BC + A\overline{B}\,\overline{C}$.

The implementation of this logic function is illustrated in Figure 5–11.



▶ FIGURE 5-11
Logic circuit for $X = \overline{A}BC + A\overline{B}\,\overline{C}$. Open file F05-11 to verify the operation.
MultiSim

EXAMPLE 5-5

Design a logic circuit to implement the operation specified in the truth table of Table 5–4.

▶ TABLE 5-4

Inputs			Output	
A	B	C	X	Product Term
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\overline{A}BC$
1	0	0	0	
1	0	1	1	$A\overline{B}C$
1	1	0	1	$AB\overline{C}$
1	1	1	0	

Solution

Notice that $X = 1$ for only three of the input conditions. Therefore, the logic expression is

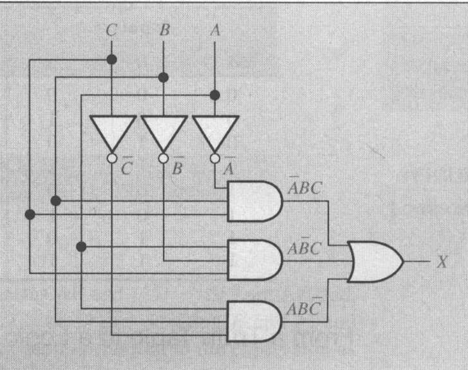
$$X = \overline{A}BC + A\overline{B}C + AB\overline{C}$$

The logic gates required are three inverters, three 3-input AND gates and one 3-input OR gate. The logic circuit is shown in Figure 5–12.

FIGURE 5-12

Open file F05-12 to verify the operation.

MultiSim



Related Problem

Determine if the logic circuit of Figure 5–12 can be simplified.

EXAMPLE 5-6

Develop a logic circuit with four input variables that will only produce a 1 output when exactly three input variables are 1s.

Solution

Out of sixteen possible combinations of four variables, the combinations in which there are exactly three 1s are listed in Table 5–5, along with the corresponding product term for each.

TABLE 5-5

A	B	C	D	Product Term
0	1	1	1	$\overline{A}BCD$
1	0	1	1	$A\overline{B}CD$
1	1	0	1	$AB\overline{C}D$
1	1	1	0	$ABC\overline{D}$

The product terms are ORed to get the following expression:

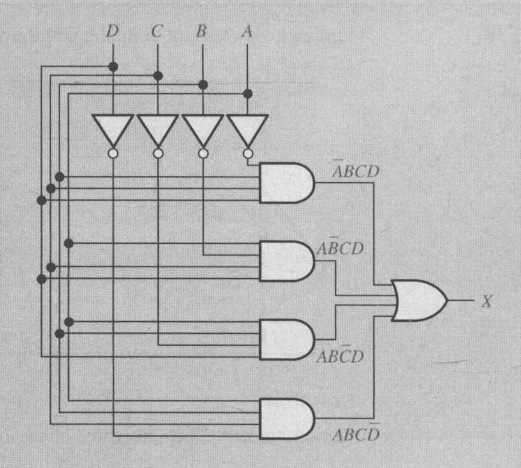
$$X = \overline{A}BCD + A\overline{B}CD + AB\overline{C}D + ABC\overline{D}$$

This expression is implemented in Figure 5–13 with AND-OR logic.

FIGURE 5-13

Open file F05-13 to verify the operation.

MultiSim



Related Problem

Determine if the logic circuit of Figure 5–13 can be simplified.

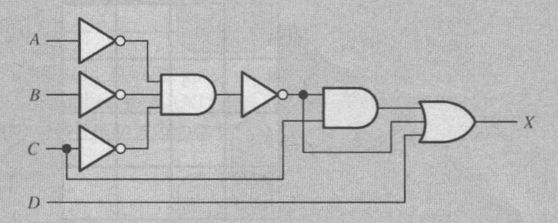
EXAMPLE 5-7

Reduce the combinational logic circuit in Figure 5–14 to a minimum form.

FIGURE 5-14

Open file F05-14 to verify that this circuit is equivalent to the gate in Figure 5–15.

MultiSim



Solution

The expression for the output of the circuit is

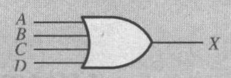
$$X = (\overline{\overline{A}\,\overline{B}\,\overline{C}})C + \overline{\overline{A}\,\overline{B}\,\overline{C}} + D$$

Applying DeMorgan's theorem and Boolean algebra,

$$\begin{aligned} X &= (\overline{\overline{A}} + \overline{\overline{B}} + \overline{\overline{C}})C + \overline{\overline{A}} + \overline{\overline{B}} + \overline{\overline{C}} + D \\ &= AC + BC + CC + A + B + C + D \\ &= AC + BC + C + A + B + \cancel{C} + D \\ &= C(A + B + 1) + A + B + D \\ X &= A + B + C + D \end{aligned}$$

The simplified circuit is a 4-input OR gate as shown in Figure 5–15.

FIGURE 5-15



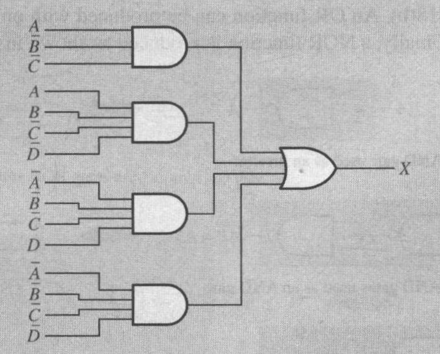
Related Problem

Verify the minimized expression $A + B + C + D$ using a Karnaugh map.

EXAMPLE 5-8

Minimize the combinational logic circuit in Figure 5–16. Inverters for the complemented variables are not shown.

FIGURE 5-16



Solution

The output expression is

$$X = A\overline{B}\,\overline{C} + AB\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$$

Expanding the first term to include the missing variables D and $\overline{D}$,

$$\begin{aligned} X &= A\overline{B}\,\overline{C}(D + \overline{D}) + AB\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} \\ &= A\overline{B}\,\overline{C}D + A\overline{B}\,\overline{C}\,\overline{D} + AB\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} \end{aligned}$$

This expanded SOP expression is mapped and simplified on the Karnaugh map in Figure 5–17(a). The simplified implementation is shown in part (b). Inverters are not shown.

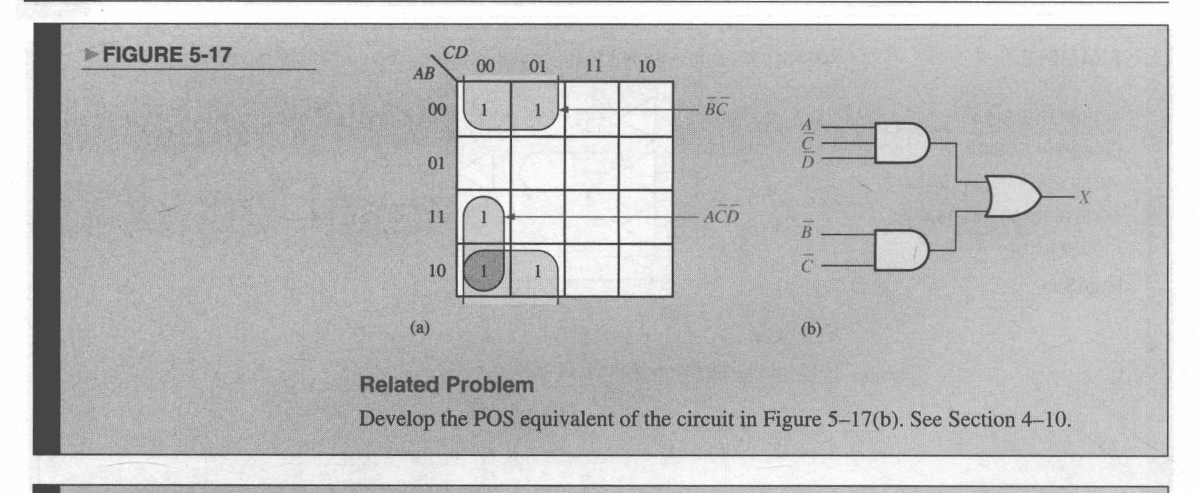


FIGURE 5-17

Related Problem

Develop the POS equivalent of the circuit in Figure 5–17(b). See Section 4–10.

SECTION 5-2 CHECKUP

1. Implement the following Boolean expressions as they are stated:

   (a) $X = ABC + AB + AC$ (b) $X = AB(C + DE)$

2. Develop a logic circuit that will produce a 1 on its output only when all three inputs are 1s or when all three inputs are 0s.
3. Reduce the circuits in Question 1 to minimum SOP form.

5-3 The Universal Property of NAND and NOR Gates

The NAND Gate as a Universal Logic Element

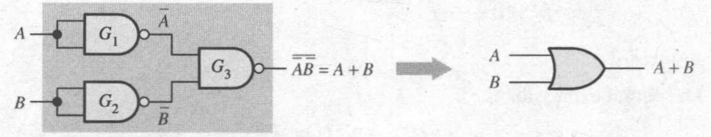
The NAND gate is a **universal gate** because it can be used to produce the NOT, the AND, the OR, and the NOR functions. An inverter can be made from a NAND gate by connecting all of the inputs together and creating, in effect, a single input, as shown in Figure 5–18(a) for a 2-input gate. An AND function can be generated by the use of NAND gates alone, as shown in Figure 5–18(b). An OR function can be produced with only NAND gates, as illustrated in part (c). Finally, a NOR function is produced as shown in part (d).

Combinations of NAND gates can be used to produce any logic function.

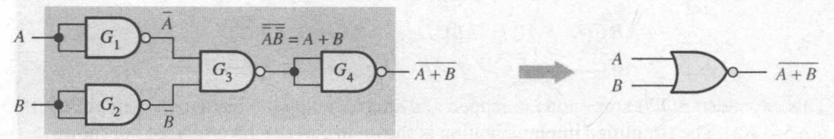
(a) One NAND gate used as an inverter

A B $\overline{AB}$ $\overline{\overline{AB}} = AB$ → A B AB

(b) Two NAND gates used as an AND gate



(c) Three NAND gates used as an OR gate



(d) Four NAND gates used as a NOR gate

FIGURE 5-18

Universal application of NAND gates. Open files F05-18(a), (b), (c), and (d) to verify each of the equivalencies.

MultiSim

In Figure 5–18(b), a NAND gate is used to invert (complement) a NAND output to form the AND function, as indicated in the following equation:

$$X = \overline{\overline{AB}} = AB$$

In Figure 5–18(c), NAND gates G_1 and G_2 are used to invert the two input variables before they are applied to NAND gate G_3. The final OR output is derived as follows by application of DeMorgan's theorem:

$$X = \overline{\overline{A}\,\overline{B}} = A + B$$

In Figure 5–18(d), NAND gate G_4 is used as an inverter connected to the circuit of part (c) to produce the NOR operation $\overline{A + B}$.

The NOR Gate as a Universal Logic Element

Combinations of NOR gates can be used to produce any logic function.

Like the NAND gate, the NOR gate can be used to produce the NOT, AND, OR, and NAND functions. A NOT circuit, or inverter, can be made from a NOR gate by connecting all of the inputs together to effectively create a single input, as shown in Figure 5–19(a) with a 2-input example. Also, an OR gate can be produced from NOR gates, as illustrated in Figure 5–19(b). An AND gate can be constructed by the use of NOR gates, as shown in Figure 5–19(c). In this case the NOR gates G_1 and G_2 are used as inverters, and the final output is derived by the use of DeMorgan's theorem as follows:

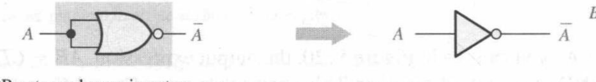
$$X = \overline{\overline{A} + \overline{B}} = AB$$

Figure 5–19(d) shows how NOR gates are used to form a NAND function.

FIGURE 5-19

Universal application of NOR gates. Open files F05-19(a), (b), (c), and (d) to verify each of the equivalencies.

MultiSim



(a) One NOR gate used as an inverter

(b) Two NOR gates used as an OR gate

(c) Three NOR gates used as an AND gate

(d) Four NOR gates used as a NAND gate

SECTION 5-3 CHECKUP

1. Use NAND gates to implement each expression:
   (a) $X = \overline{A} + B$ (b) $X = A\overline{B}$
2. Use NOR gates to implement each expression:
   (a) $X = \overline{A} + B$ (b) $X = A\overline{B}$

5-4 Combinational Logic Using NAND and NOR Gates

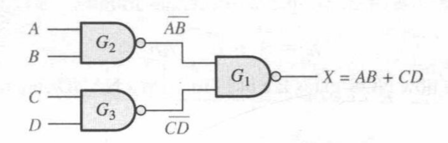
NAND Logic

As you have learned, a NAND gate can function as either a NAND or a negative-OR because, by DeMorgan's theorem,

$$\underset{\text{NAND}}{\overline{AB}} = \underset{\text{negative-OR}}{\overline{A} + \overline{B}}$$

Consider the NAND logic in Figure 5–20. The output expression is developed in the following steps:

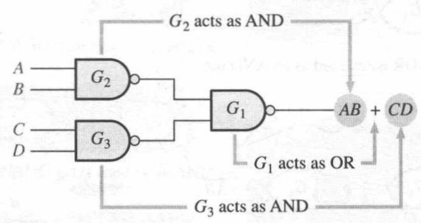
$$\begin{aligned} X &= \overline{(\overline{AB})(\overline{CD})} \\ &= \overline{(\overline{A} + \overline{B})(\overline{C} + \overline{D})} \\ &= \overline{(\overline{A} + \overline{B})} + \overline{(\overline{C} + \overline{D})} \\ &= \overline{\overline{A}}\,\overline{\overline{B}} + \overline{\overline{C}}\,\overline{\overline{D}} \\ &= AB + CD \end{aligned}$$



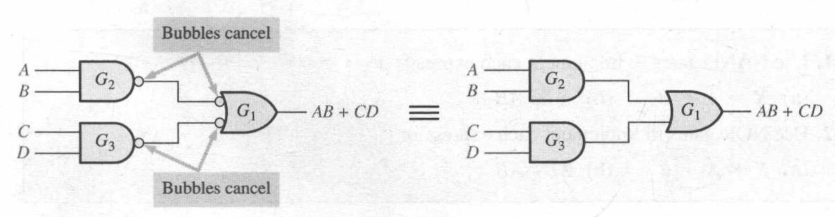
◄ **FIGURE 5-20**
NAND logic for $X = AB + CD$.

As you can see in Figure 5–20, the output expression, $AB + CD$, is in the form of two AND terms ORed together. This shows that gates G_2 and G_3 act as AND gates and that gate G_1 acts as an OR gate, as illustrated in Figure 5–21(a). This circuit is redrawn in part (b) with NAND symbols for gates G_2 and G_3 and a negative-OR symbol for gate G_1.

Notice in Figure 5–21(b) the bubble-to-bubble connections between the outputs of gates G_2 and G_3 and the inputs of gate G_1. *Since a bubble represents an inversion, two connected bubbles represent a double inversion and therefore cancel each other.* This inversion cancellation can be seen in the previous development of the output expression $AB + CD$ and is indicated by the absence of barred terms in the output expression. Thus, the circuit in Figure 5–21(b) is *effectively* an AND-OR circuit, as shown in Figure 5–21(c).



(a) Original NAND logic diagram showing effective gate operation relative to the output expression



(b) Equivalent NAND/Negative-OR logic diagram (c) AND-OR equivalent

◄ **FIGURE 5-21**
Development of the AND-OR equivalent of the circuit in Figure 5–20.

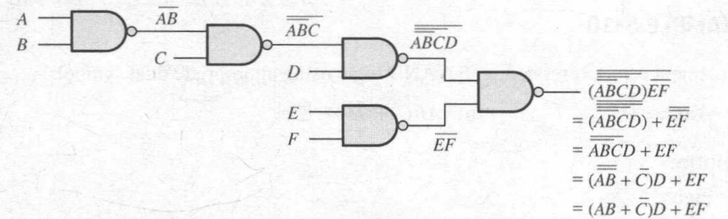
NAND Logic Diagrams Using Dual Symbols

All logic diagrams using NAND gates should be drawn with each gate represented by either a NAND symbol or the equivalent negative-OR symbol to reflect the operation of the gate within the logic circuit. The NAND symbol and the **negative-OR** symbol are called *dual symbols.* When drawing a NAND logic diagram, always use the gate symbols in such a way that every connection between a gate output and a gate input is either bubble-to-bubble or nonbubble-to-nonbubble. In general, a bubble output should not be connected to a nonbubble input or vice versa in a logic diagram.

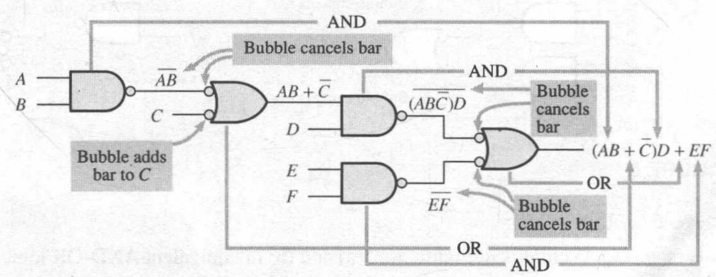
Figure 5–22 shows an arrangement of gates to illustrate the procedure of using the appropriate dual symbols for a NAND circuit with several gate levels. Although using all NAND symbols as in Figure 5–22(a) is correct, the diagram in part (b) is much easier to "read" and is the preferred method. As shown in Figure 5–22(b), the output gate is represented with a negative-OR symbol. Then the NAND symbol is used for the level of gates right before the output gate and the symbols for successive levels of gates are alternated as you move away from the output.

▶ **FIGURE 5-22**

Illustration of the use of the appropriate dual symbols in a NAND logic diagram.



(a) Several Boolean steps are required to arrive at final output expression.

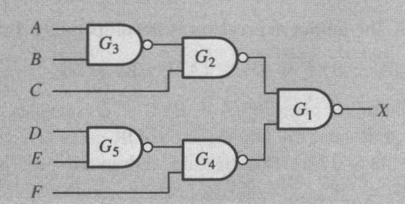


(b) Output expression can be obtained directly from the function of each gate symbol in the diagram.

The shape of the gate indicates the way its inputs will appear in the output expression and thus shows how the gate functions within the logic circuit. For a NAND symbol, the inputs appear ANDed in the output expression; and for a negative-OR symbol, the inputs appear ORed in the output expression, as Figure 5–22(b) illustrates. The dual-symbol diagram in part (b) makes it easier to determine the output expression directly from the logic diagram because each gate symbol indicates the relationship of its input variables as they appear in the output expression.

EXAMPLE 5-9

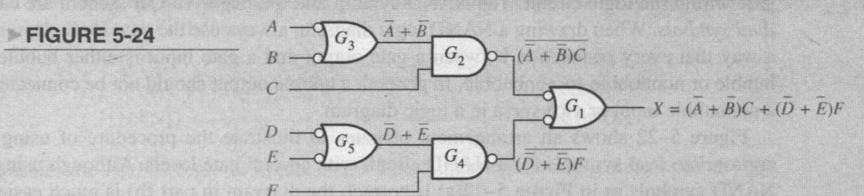
Redraw the logic diagram and develop the output expression for the circuit in Figure 5–23 using the appropriate dual symbols.



▶ **FIGURE 5-23**

Solution

Redraw the logic diagram in Figure 5–23 with the use of equivalent negative-OR symbols as shown in Figure 5–24. Writing the expression for X directly from the indicated logic operation of each gate gives $X = (\overline{A} + \overline{B})C + (\overline{D} + \overline{E})F$.



▶ **FIGURE 5-24**

Related Problem

Derive the output expression from Figure 5–23 and show it is equivalent to the expression in the solution.

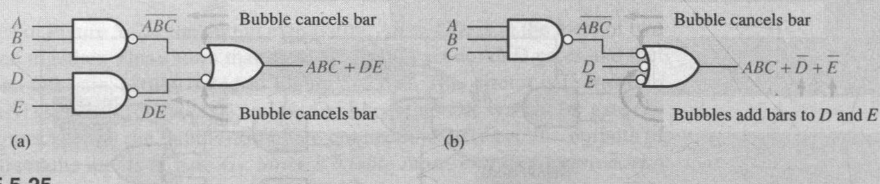
EXAMPLE 5-10

Implement each expression with NAND logic using appropriate dual symbols:

(a) $ABC + DE$ **(b)** $ABC + \overline{D} + \overline{E}$

Solution

See Figure 5–25.



▲ **FIGURE 5-25**

Related Problem

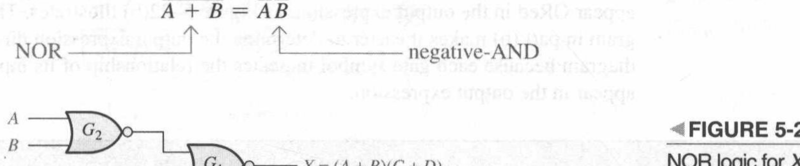
Convert the NAND circuits in Figure 5–25(a) and (b) to equivalent AND-OR logic.

NOR Logic

A NOR gate can function as either a NOR or a **negative-AND**, as shown by DeMorgan's theorem.

$$\overline{A + B} = \overline{A}\,\overline{B}$$

NOR ↑ ↑ negative-AND



◀ **FIGURE 5-26**

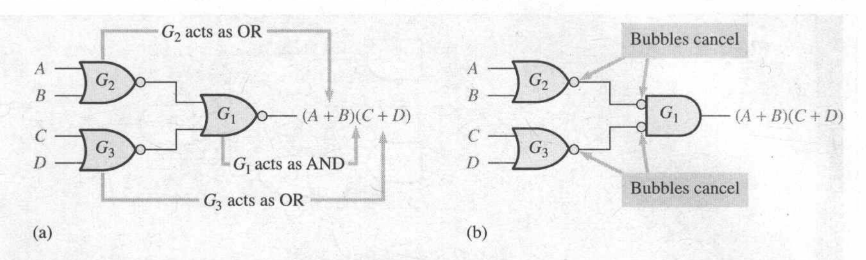
NOR logic for $X = (A + B)(C + D)$.

Consider the NOR logic in Figure 5–26. The output expression is developed as follows:

$$X = \overline{\overline{A + B} + \overline{C + D}} = (\overline{\overline{A + B}})(\overline{\overline{C + D}}) = (A + B)(C + D)$$

As you can see in Figure 5–26, the output expression $(A + B)(C + D)$ consists of two OR terms ANDed together. This shows that gates G_2 and G_3 act as OR gates and gate G_1 acts as an AND gate, as illustrated in Figure 5–27(a). This circuit is redrawn in part (b) with a negative-AND symbol for gate G_1.

▶ FIGURE 5-27

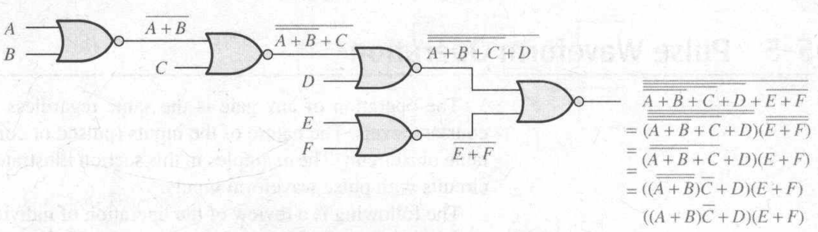


NOR Logic Diagram Using Dual Symbols

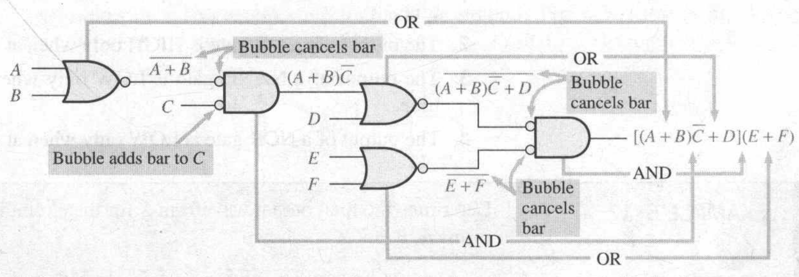
As with NAND logic, the purpose for using the dual symbols is to make the logic diagram easier to read and analyze, as illustrated in the NOR logic circuit in Figure 5–28. When the circuit in part (a) is redrawn with dual symbols in part (b), notice that all output-to-input connections between gates are bubble-to-bubble or nonbubble-to-nonbubble. Again, you can see that the shape of each gate symbol indicates the type of term (AND or OR) that it produces in the output expression, thus making the output expression easier to determine and the logic diagram easier to analyze.

▶ FIGURE 5-28

Illustration of the use of the appropriate dual symbols in a NOR logic diagram.



(a) Final output expression is obtained after several Boolean steps.

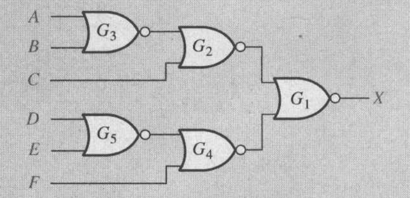


(b) Output expression can be obtained directly from the function of each gate symbol in the diagram.

EXAMPLE 5-11

Using appropriate dual symbols, redraw the logic diagram and develop the output expression for the circuit in Figure 5–29.

▶ FIGURE 5-29

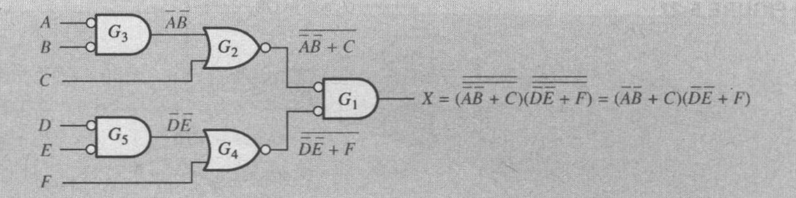


Solution

Redraw the logic diagram with the equivalent negative-AND symbols as shown in Figure 5–30. Writing the expression for X directly from the indicated operation of each gate,

$$X = (\overline{A}\,\overline{B} + C)(\overline{D}\,\overline{E} + F)$$

▶ FIGURE 5-30



Related Problem

Prove that the output of the NOR circuit in Figure 5–29 is the same as for the circuit in Figure 5–30.

SECTION 5-4 CHECKUP

1. Implement the expression $X = \overline{(\overline{A} + \overline{B} + \overline{C})DE}$ by using NAND logic.
2. Implement the expression $X = \overline{\overline{A}\,\overline{B}\,\overline{C} + (D + E)}$ with NOR logic.

5-5 Pulse Waveform Operation

The operation of any gate is the same regardless of whether its inputs are pulsed or constant levels. The nature of the inputs (pulsed or constant levels) does not alter the truth table of a circuit. The examples in this section illustrate the analysis of combinational logic circuits with pulse waveform inputs.

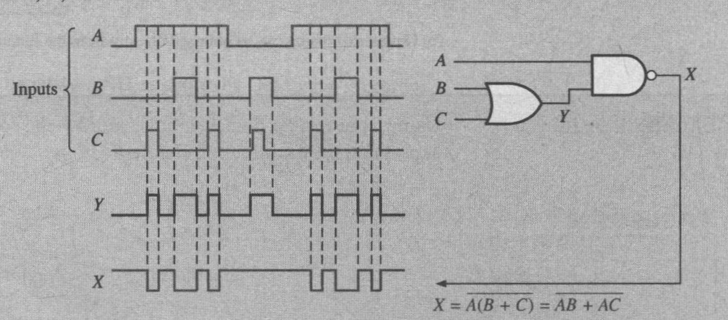
The following is a review of the operation of individual gates for use in analyzing combinational circuits with pulse waveform inputs:

1. The output of an AND gate is HIGH only when all inputs are HIGH at the same time.
2. The output of an OR gate is HIGH only when at least one of its inputs is HIGH.
3. The output of a NAND gate is LOW only when all inputs are HIGH at the same time.
4. The output of a NOR gate is LOW only when at least one of its inputs is HIGH.

EXAMPLE 5-12

Determine the final output waveform X for the circuit in Figure 5–31, with input waveforms A, B, and C as shown.

▶ FIGURE 5-31



Solution

The output expression, $\overline{AB + AC}$, indicates that the output X is LOW when both A and B are HIGH or when both A and C are HIGH or when all inputs are HIGH. The output waveform X is shown in the timing diagram of Figure 5–31. The intermediate waveform Y at the output of the OR gate is also shown.

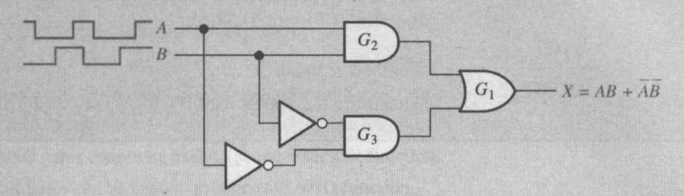
Related Problem

Determine the output waveform if input A is a constant HIGH level.

EXAMPLE 5-13

Draw the timing diagram for the circuit in Figure 5–32 showing the outputs of G_1, G_2, and G_3 with the input waveforms, A, and B, as indicated.

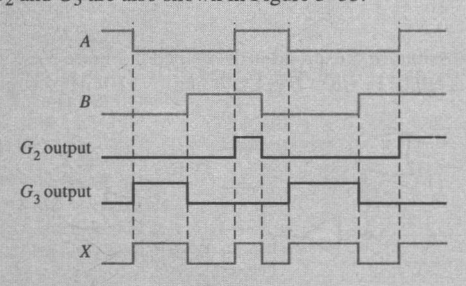
▶ FIGURE 5-32



Solution

When both inputs are HIGH or when both inputs are LOW, the output X is HIGH as shown in Figure 5–33. Notice that this is an exclusive-NOR circuit. The intermediate outputs of gates G_2 and G_3 are also shown in Figure 5–33.

▶ FIGURE 5-33



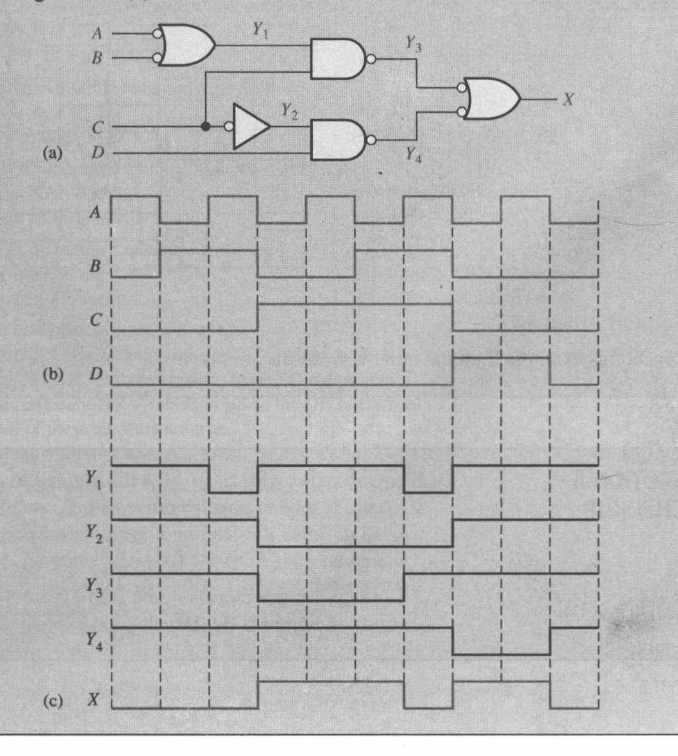
Related Problem

Determine the output X in Figure 5–32 if input B is inverted.

EXAMPLE 5-14

Determine the output waveform X for the logic circuit in Figure 5–34(a) by first finding the intermediate waveform at each of points Y_1, Y_2, Y_3, and Y_4. The input waveforms are shown in Figure 5–34(b).

▶ FIGURE 5-34



Solution

All the intermediate waveforms and the final output waveform are shown in the timing diagram of Figure 5–34(c).

Related Problem

Determine the waveforms Y_1, Y_2, Y_3, Y_4 and X if input waveform A is inverted.

EXAMPLE 5-15

Determine the output waveform X for the circuit in Example 5–14, Figure 5–34(a), directly from the output expression.

Solution

The output expression for the circuit is developed in Figure 5–35. The SOP form indicates that the output is HIGH when A is LOW and C is HIGH or when B is LOW and C is HIGH or when C is LOW and D is HIGH.

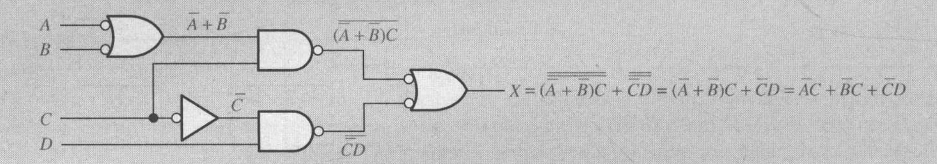
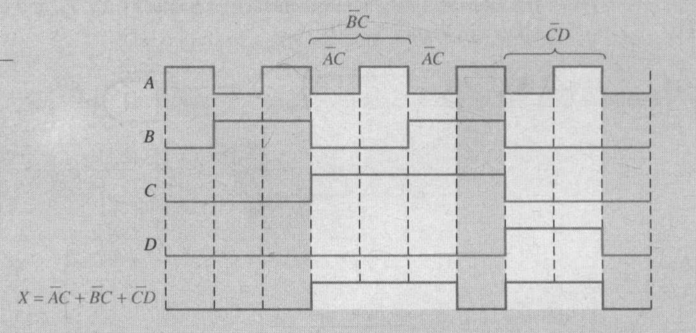


FIGURE 5-35

The result is shown in Figure 5–36 and is the same as the one obtained by the intermediate-waveform method in Example 5–14. The corresponding product terms for each waveform condition that results in a HIGH output are indicated.

FIGURE 5-36



Related Problem

Repeat this example if all the input waveforms are inverted.

SECTION 5-5 CHECKUP

1. One pulse with $t_W = 50\ \mu s$ is applied to one of the inputs of an exclusive-OR circuit. A second positive pulse with $t_W = 10\ \mu s$ is applied to the other input beginning $15\ \mu s$ after the leading edge of the first pulse. Show the output in relation to the inputs.
2. The pulse waveforms A and B in Figure 5–31 are applied to the exclusive-NOR circuit in Figure 5–32. Develop a complete timing diagram.

TRUE/FALSE QUIZ

Answers are at the end of the chapter.

1. AND-OR logic can have only two 2-input AND gates.
2. AOI is an acronym for AND-OR-Invert.
3. If the inputs of an exclusive-OR gate are the same, the output is HIGH (1).
4. If the inputs of an exclusive-NOR gate are different, the output is LOW (0).
5. A parity generator can be implemented using exclusive-OR gates.
6. NAND gates cannot be used to produce the OR function.
7. NOR gates can be used to produce the AND function.
8. Any SOP expression can be implemented using only NAND gates.
9. The dual symbol for a NAND gate is a negative-AND symbol.
10. Negative-OR is equivalent to NAND.

SELF-TEST

Answers are at the end of the chapter.

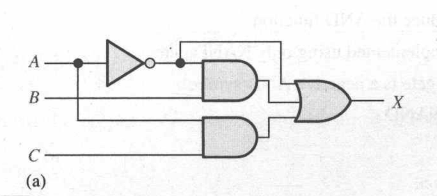
1. The output expression for an AND-OR circuit having one AND gate with inputs A, B, C, and D and one AND gate with inputs E and F is
   (a) $ABCDEF$ **(b)** $A + B + C + D + E + F$
   (c) $(A + B + C + D)(E + F)$ **(d)** $ABCD + EF$
2. A logic circuit with an output $X = A\overline{B}C + A\overline{C}$ consists of
   (a) two AND gates and one OR gate
   (b) two AND gates, one OR gate, and two inverters
   (c) two OR gates, one AND gate, and two inverters
   (d) two AND gates, one OR gate, and one inverter
3. To implement the expression $\overline{A}BCD + A\overline{B}CD + AB\overline{C}\,\overline{D}$, it takes one OR gate and
   (a) one AND gate **(b)** three AND gates
   (c) three AND gates and four inverters **(d)** three AND gates and three inverters
4. The expression $\overline{A}BCD + ABC\overline{D} + A\overline{B}\,\overline{C}D$
   (a) cannot be simplified **(b)** can be simplified to $\overline{A}BC + A\overline{B}$
   (c) can be simplified to $ABC\overline{D} + \overline{A}B\overline{C}$ **(d)** None of these answers is correct.
5. The output expression for an AND-OR-Invert circuit having one AND gate with inputs, A, B, C, and D and one AND gate with inputs E and F is
   (a) $ABCD + EF$ **(b)** $\overline{A} + \overline{B} + \overline{C} + \overline{D} + \overline{E} + \overline{F}$
   (c) $\overline{(A + B + C + D)(E + F)}$ **(d)** $(\overline{A} + \overline{B} + \overline{C} + \overline{D})(\overline{E} + \overline{F})$
6. An exclusive-OR function is expressed as
   (a) $\overline{A}\,\overline{B} + AB$ **(b)** $\overline{A}B + A\overline{B}$
   (c) $(\overline{A} + B)(A + \overline{B})$ **(d)** $(\overline{A} + \overline{B}) + (A + B)$
7. The AND operation can be produced with
   (a) two NAND gates **(b)** three NAND gates
   (c) one NOR gate **(d)** three NOR gates
8. The OR operation can be produced with
   (a) two NOR gates **(b)** three NAND gates
   (c) four NAND gates **(d)** both answers (a) and (b)
9. When using dual symbols in a logic diagram,
   (a) bubble outputs are connected to bubble inputs
   (b) the NAND symbols produce the AND operations
   (c) the negative-OR symbols produce the OR operations
   (d) All of these answers are true.
   (e) None of these answers is true.
10. All Boolean expressions can be implemented with
    (a) NAND gates only
    (b) NOR gates only
    (c) combinations of NAND and NOR gates
    (d) combinations of AND gates, OR gates, and inverters
    (e) any of these

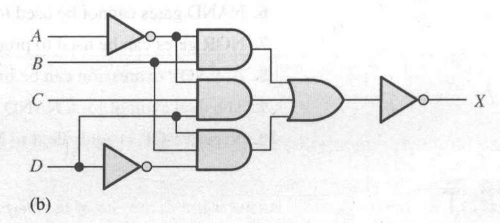
PROBLEMS

Answers to odd-numbered problems are at the end of the book.

Section 5–1 Basic Combinational Logic Circuits

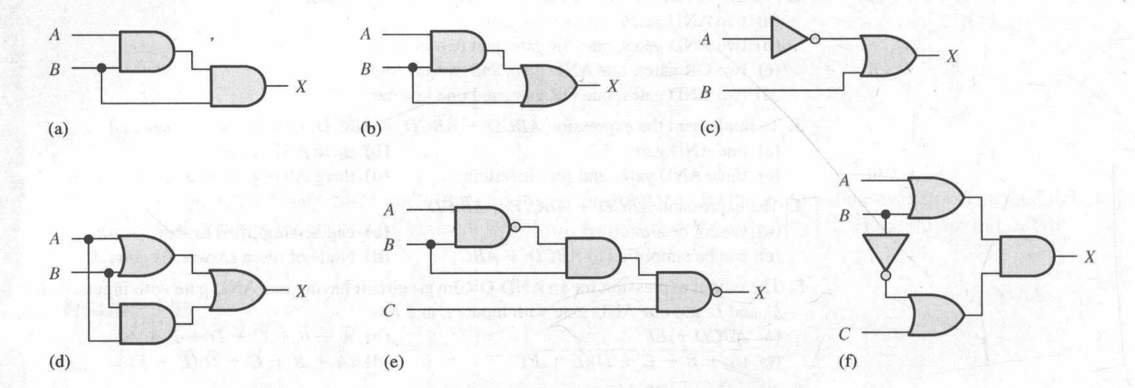
1. Draw the ANSI distinctive shape logic diagram for a 3-wide, 4-input AND-OR-Invert circuit. Also draw the ANSI standard rectangular outline symbol.
2. Write the output expression for each circuit in Figure 5–37.





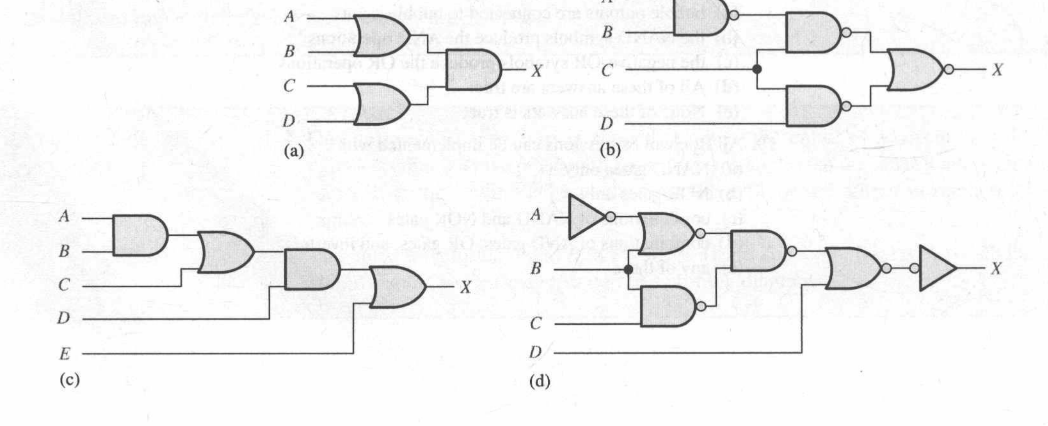
▲ **FIGURE 5-37**

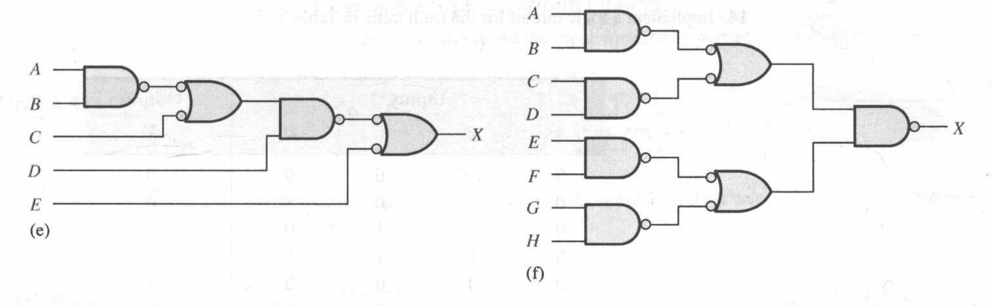
3. Write the output expression for each circuit as it appears in Figure 5–38.



▲ **FIGURE 5-38**

4. Write the output expression for each circuit as it appears in Figure 5–39 and then change each circuit to an equivalent AND-OR configuration.
5. Develop the truth table for each circuit in Figure 5–38.
6. Develop the truth table for each circuit in Figure 5–39.
7. Show that an exclusive-NOR circuit produces a POS output.





▲ **FIGURE 5-39**

Section 5–2 Implementing Combinational Logic

8. Develop an AND-OR-Invert logic circuit for a power saw that removes power (logic 0) if the guard is not in place (logic 0) and the switch is *on* (logic 1) or the switch is *on* and the motor is too hot (logic 1).

9. An AOI (AND-OR-Invert) logic chip has two 4-input AND gates connected to a 2-input NOR gate. Write the Boolean expression for the circuit (assume the inputs are labeled *A* through *H*).

10. Use AND gates, OR gates, or combinations of both to implement the following logic expressions as stated:

(a) $X = AB$
(b) $X = A + B$
(c) $X = AB + C$
(d) $X = ABC + D$
(e) $X = A + B + C$
(f) $X = ABCD$
(g) $X = A(CD + B)$
(h) $X = AB(C + DEF) + CE(A + B + F)$

11. Use AND gates, OR gates, and inverters as needed to implement the following logic expressions as stated:

(a) $X = AB + \overline{B}C$
(b) $X = A(B + \overline{C})$
(c) $X = A\overline{B} + AB$
(d) $X = \overline{ABC} + B(EF + \overline{G})$
(e) $X = A[BC(A + B + C + D)]$
(f) $X = B(C\overline{D}E + \overline{E}FG)(\overline{AB} + C)$

12. Use NAND gates, NOR gates, or combinations of both to implement the following logic expressions as stated:

(a) $X = \overline{A}B + CD + (\overline{A + B})(ACD + \overline{BE})$
(b) $X = AB\overline{C}\overline{D} + D\overline{E}F + \overline{AF}$
(c) $X = \overline{A}[B + \overline{C}(D + E)]$

13. Implement a logic circuit for the truth table in Table 5–6.

► **TABLE 5-6**

Inputs			Output
A	*B*	*C*	*X*
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

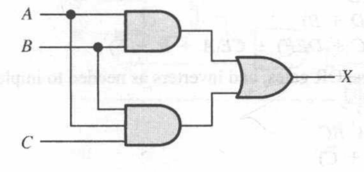
14. Implement a logic circuit for the truth table in Table 5–7.

▶ **TABLE 5-7**

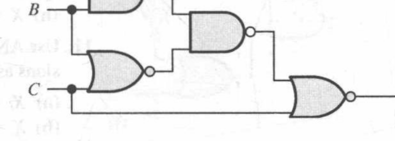
Inputs				Output
A	*B*	*C*	*D*	*X*
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

15. Simplify the circuit in Figure 5–40 as much as possible, and verify that the simplified circuit is equivalent to the original by showing that the truth tables are identical.

16. Repeat Problem 15 for the circuit in Figure 5–41.



▲ **FIGURE 5-40**



▲ **FIGURE 5-41**

17. Minimize the gates required to implement the functions in each part of Problem 11 in SOP form.

18. Minimize the gates required to implement the functions in each part of Problem 12 in SOP form.

19. Minimize the gates required to implement the function of the circuit in each part of Figure 5–39 in SOP form.

Section 5–3 The Universal Property of NAND and NOR Gates

20. Implement the logic circuits in Figure 5–37 using only NAND gates.

21. Implement the logic circuit in Figure 5–41 using only NAND gates.

22. Repeat Problem 20 using only NOR gates.

23. Repeat Problem 21 using only NOR gates.

Section 5–4 Combinational Logic Using NAND and NOR Gates

24. Show how the following expressions can be implemented as stated using only NOR gates:

(a) $X = ABC$ **(b)** $X = \overline{ABC}$ **(c)** $X = A + B$

(d) $X = A + B + \overline{C}$ **(e)** $X = \overline{AB} + \overline{CD}$ **(f)** $X = (A + B)(C + D)$

(g) $X = AB[C(\overline{DE} + \overline{AB}) + \overline{BCE}]$

25. Repeat Problem 24 using only NAND gates.

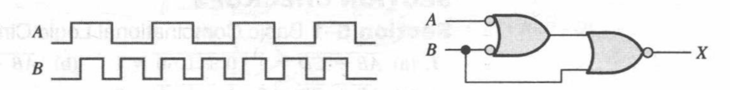
26. Implement each function in Problem 10 by using only NAND gates.

27. Implement each function in Problem 11 by using only NAND gates.

Section 5–5 Pulse Waveform Operation

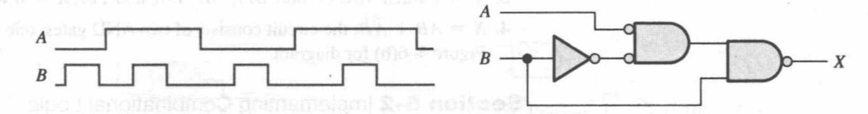
28. Given the logic circuit and the input waveforms in Figure 5–42, draw the output waveform.

▶ FIGURE 5-42



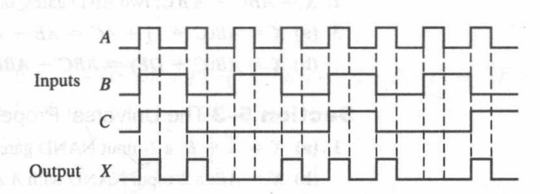
29. For the logic circuit in Figure 5–43, draw the output waveform in proper relationship to the inputs.

▶ FIGURE 5-43



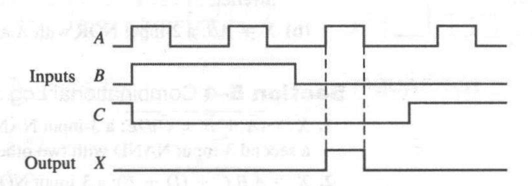
30. For the input waveforms in Figure 5–44, what logic circuit will generate the output waveform shown?

▶ FIGURE 5-44



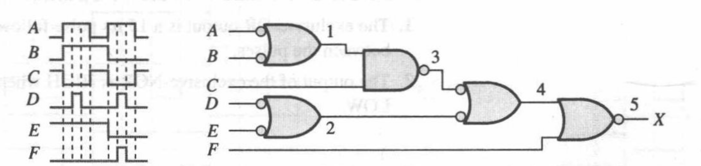
31. Repeat Problem 30 for the waveforms in Figure 5–45.

▶ FIGURE 5-45

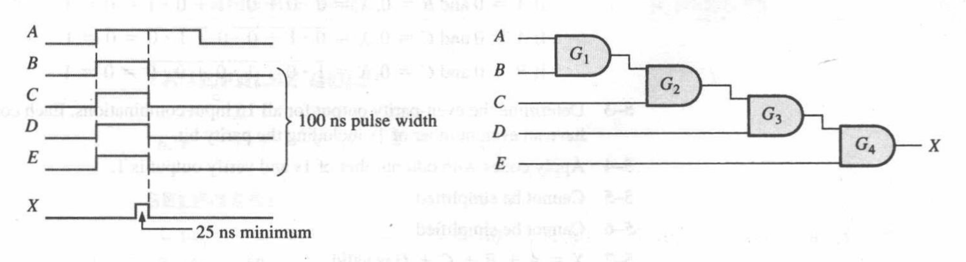


32. For the circuit in Figure 5–46, draw the waveforms at the numbered points in the proper relationship to each other.

▶ FIGURE 5-46



33. Assuming a propagation delay through each gate of 10 nanoseconds (ns), determine if the *desired* output waveform X in Figure 5–47 (a pulse with a minimum $t_W = 25$ ns positioned as shown) will be generated properly with the given inputs.



▲ FIGURE 5-47

ANSWERS

SECTION CHECKUPS

Section 5–1 Basic Combinational Logic Circuits

1. (a) $\overline{AB + CD} = \overline{1 \cdot 0 + 1 \cdot 0} = 1$ (b) $\overline{AB + CD} = \overline{1 \cdot 1 + 0 \cdot 1} = 0$
   (c) $\overline{AB + CD} = \overline{0 \cdot 1 + 1 \cdot 1} = 0$
2. (a) $A\overline{B} + \overline{A}B = 1 \cdot \overline{0} + \overline{1} \cdot 0 = 1$ (b) $A\overline{B} + \overline{A}B = 1 \cdot \overline{1} + \overline{1} \cdot 1 = 0$
   (c) $A\overline{B} + \overline{A}B = 0 \cdot \overline{1} + \overline{0} \cdot 1 = 1$ (d) $A\overline{B} + \overline{A}B = 0 \cdot \overline{0} + \overline{0} \cdot 0 = 0$
3. $X = 1$ when $ABC = 000, 011, 101, 110$, and 111; $X = 0$ when $ABC = 001, 010$, and 100
4. $X = AB + \overline{A}\,\overline{B}$; the circuit consists of two AND gates, one OR gate, and two inverters. See Figure 5–6(b) for diagram.

Section 5–2 Implementing Combinational Logic

1. (a) $X = ABC + AB + AC$: three AND gates, one OR gate
   (b) $X = AB(C + DE)$: three AND gates, one OR gate
2. $X = ABC + \overline{A}\,\overline{B}\,\overline{C}$; two AND gates, one OR gate, and three inverters
3. (a) $X = AB(C + 1) + AC = AB + AC$
   (b) $X = AB(C + DE) = ABC + ABDE$

Section 5–3 The Universal Property of NAND and NOR Gates

1. (a) $X = \overline{A} + B$: a 2-input NAND gate with A and $\overline{B}$ on its inputs.
   (b) $X = A\overline{B}$: a 2-input NAND with A and $\overline{B}$ on its inputs, followed by one NAND used as an inverter.
2. (a) $X = \overline{A} + B$: a 2-input NOR with inputs $\overline{A}$ and B, followed by one NOR used as an inverter.
   (b) $X = A\overline{B}$: a 2-input NOR with $\overline{A}$ and B on its inputs.

Section 5–4 Combinational Logic Using NAND and NOR Gates

1. $X = \overline{(\overline{A} + \overline{B} + \overline{C})DE}$: a 3-input NAND with inputs, A, B, and C, with its output connected to a second 3-input NAND with two other inputs, D and E
2. $X = \overline{\overline{A}\,\overline{B}\,\overline{C} + (D + E)}$: a 3-input NOR with inputs A, B, and C, with its output connected to a second 3-input NOR with two other inputs, D and E

Section 5–5 Pulse Waveform Operation

1. The exclusive-OR output is a 15 μs pulse followed by a 25 μs pulse, with a separation of 10 μs between the pulses.
2. The output of the exclusive-NOR is HIGH when both inputs are HIGH or when both inputs are LOW.

RELATED PROBLEMS FOR EXAMPLES

5–1 $X = AB + AC + BC$

5–2 $X = \overline{AB + AC + BC}$

If $A = 0$ and $B = 0$, $X = \overline{0 \cdot 0 + 0 \cdot 1 + 0 \cdot 1} = \overline{0} = 1$

If $A = 0$ and $C = 0$, $X = \overline{0 \cdot 1 + 0 \cdot 0 + 1 \cdot 0} = \overline{0} = 1$

If $B = 0$ and $C = 0$, $X = \overline{1 \cdot 0 + 1 \cdot 0 + 0 \cdot 0} = \overline{0} = 1$

5–3 Determine the even-parity output for all 16 input combinations. Each combination should have an even number of 1s including the parity bit.

5–4 Apply codes with odd number of 1s and verify output is 1.

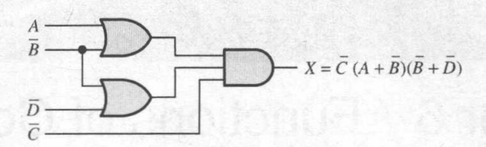
5–5 Cannot be simplified

5–6 Cannot be simplified

5–7 $X = A + B + C + D$ is valid.

5–8 See Figure 5–48.

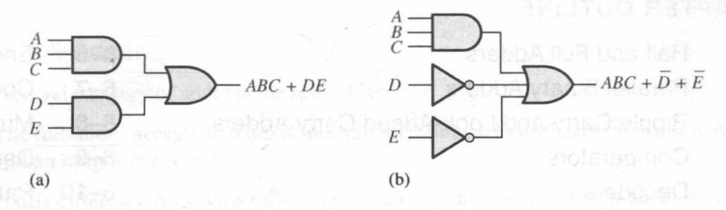
► **FIGURE 5-48**



5–9 $X = \overline{(\overline{\overline{AB}C})(\overline{\overline{DE}F})} = (\overline{AB})C + (\overline{DE})F = (\overline{A} + \overline{B})C + (\overline{D} + \overline{E})F$

5–10 See Figure 5–49.

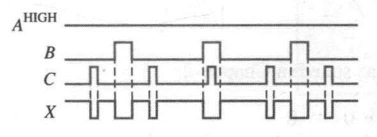
► **FIGURE 5-49**



5–11 $X = \overline{(\overline{A + B} + C) + (\overline{D + E} + F)} = (\overline{A + B} + C)(\overline{D + E} + F) = (\overline{A}\,\overline{B} + C)(\overline{D}\,\overline{E} + F)$

5–12 See Figure 5–50.

5–13 See Figure 5–51.

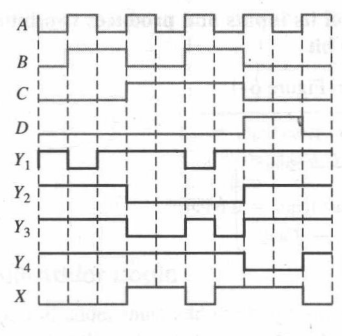


▲ **FIGURE 5-50**

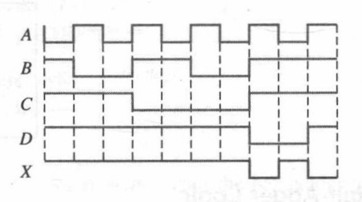
▲ **FIGURE 5-51**

5–14 See Figure 5–52.

5–15 See Figure 5–53.



▲ **FIGURE 5-52**



▲ **FIGURE 5-53**

TRUE/FALSE QUIZ

1. F **2.** T **3.** F **4.** T **5.** T
6. F **7.** T **8.** T **9.** F **10.** T

SELF-TEST

1. (d) **2.** (b) **3.** (c) **4.** (a) **5.** (d) **6.** (b) **7.** (a) **8.** (d)
9. (d) **10.** (e)

Chapter 6 Functions of Combinational Logic

CHAPTER OUTLINE

6-1 Half and Full Adders

The Half-Adder

A half-adder adds two bits and produces a sum and an output carry.

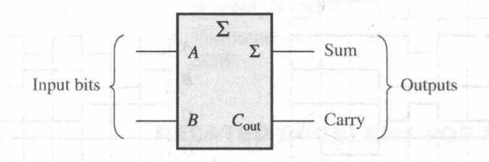
Recall the basic rules for binary addition as stated in Chapter 2.

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 10$$

The operations are performed by a logic circuit called a **half-adder**.

> **The half-adder accepts two binary digits on its inputs and produces two binary digits on its outputs—a sum bit and a carry bit.**

A half-adder is represented by the logic symbol in Figure 6–1.



MultiSim

◀ **FIGURE 6-1**

Logic symbol for a half-adder. Open file F06-01 to verify operation. *A multisim tutorial is available on the website.*

Half-Adder Logic

From the operation of the half-adder as stated in Table 6–1, expressions can be derived for the sum and the output carry as functions of the inputs. Notice that the output carry (C_{out}) is a 1 only when both A and B are 1s; therefore, C_{out} can be expressed as the AND of the input variables.

$$C_{out} = AB \qquad \text{Equation 6–1}$$

Now observe that the sum output (Σ) is a 1 only if the input variables, A and B, are not equal. The sum can therefore be expressed as the exclusive-OR of the input variables.

$$\Sigma = A \oplus B \qquad \text{Equation 6–2}$$

▼ **TABLE 6-1**

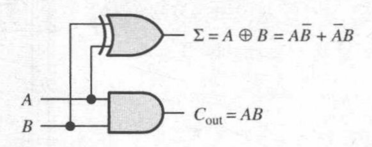
Half-adder truth table.

A	B	C_{out}	Σ
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Σ = sum
C_{out} = output carry
A and B = input variables (operands)

From Equations 6–1 and 6–2, the logic implementation required for the half-adder function can be developed. The output carry is produced with an AND gate with A and B on the inputs, and the sum output is generated with an exclusive-OR gate, as shown in Figure 6–2. Remember that the exclusive-OR can be implemented with AND gates, an OR gate, and inverters.

▶ **FIGURE 6-2**
Half-adder logic diagram.



The Full-Adder

A full-adder has an input carry while the half-adder does not.

The second category of adder is the **full-adder**.

> **The full-adder accepts two input bits and an input carry and generates a sum output and an output carry.**

The basic difference between a full-adder and a half-adder is that the full-adder accepts an input carry. A logic symbol for a full-adder is shown in Figure 6–3, and the truth table in Table 6–2 shows the operation of a full-adder.

▶ **FIGURE 6-3**
Logic symbol for a full-adder. Open file F06-03 to verify operation. MultiSim

Σ
Input bits: A, B
Input carry: C_{in}
Σ — Sum
C_{out} — Output carry

▶ **TABLE 6-2**
Full-adder truth table.

A	B	C_{in}	C_{out}	Σ
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

C_{in} = input carry, sometimes designated as CI
C_{out} = output carry, sometimes designated as CO
Σ = sum
A and B = input variables (operands)

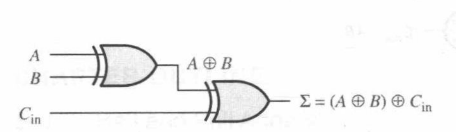
Full-Adder Logic

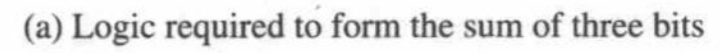
The full-adder must add the two input bits and the input carry. From the half-adder you know that the sum of the input bits A and B is the exclusive-OR of those two variables, $A \oplus B$. For the input carry (C_{in}) to be added to the input bits, it must be exclusive-ORed with $A \oplus B$, yielding the equation for the sum output of the full-adder.

$$\Sigma = (A \oplus B) \oplus C_{in} \qquad \text{Equation 6–3}$$

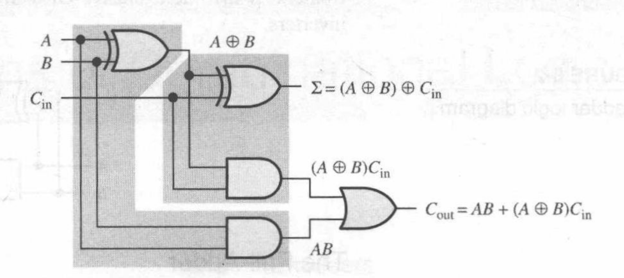
ent the full-adder sum function, two 2-input exclusive-OR gates
t generate the term $A \oplus B$, and the second has as its inputs the
e and the input carry, as illustrated in Figure 6–4(a).

This means that to implement the full-adder sum function, two 2-input exclusive-OR gates can be used. The first must generate the term $A \oplus B$, and the second has as its inputs the output of the first XOR gate and the input carry, as illustrated in Figure 6–4(a).





(a) Logic required to form the sum of three bits



(b) Complete logic circuit for a full-adder (each half-adder is enclosed by a shaded area)

▲ **FIGURE 6-4**

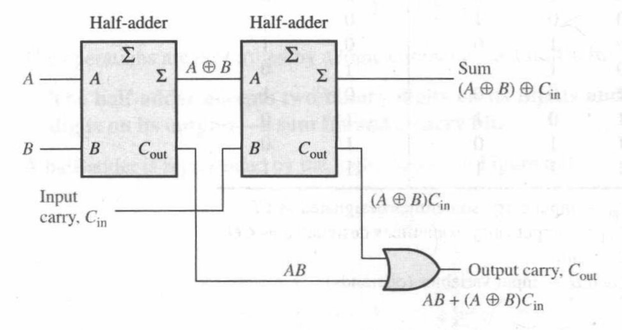
Full-adder logic. Open file F06-04 to verify operation.

MultiSim

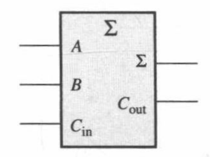
The output carry is a 1 when both inputs to the first XOR gate are 1s or when both inputs to the second XOR gate are 1s. You can verify this fact by studying Table 6–2. The output carry of the full-adder is therefore produced by input A ANDed with input B and $A \oplus B$ ANDed with C_{in}. These two terms are ORed, as expressed in Equation 6–4. This function is implemented and combined with the sum logic to form a complete full-adder circuit, as shown in Figure 6–4(b).

$$C_{out} = AB + (A \oplus B)C_{in} \qquad \text{Equation 6–4}$$

Notice in Figure 6–4(b) there are two half-adders, connected as shown in the block diagram of Figure 6–5(a), with their output carries ORed. The logic symbol shown in Figure 6–5(b) will normally be used to represent the full-adder.



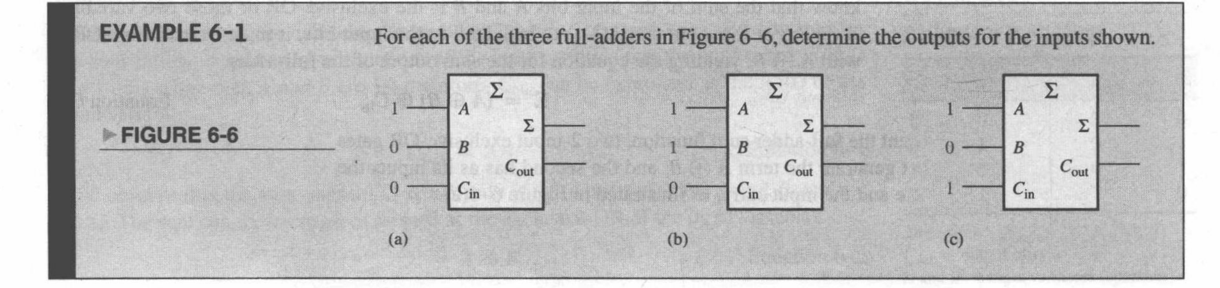
(a) Arrangement of two half-adders to form a full-adder



(b) Full-adder logic symbol

◀ **FIGURE 6-5**

Full-adder implemented with half-adders.



Solution

(a) The input bits are $A = 1, B = 0$, and $C_{in} = 0$.

$$1 + 0 + 0 = 1 \text{ with no carry}$$

Therefore, $\Sigma = \mathbf{1}$ and $C_{out} = \mathbf{0}$.

(b) The input bits are $A = 1, B = 1$, and $C_{in} = 0$.

$$1 + 1 + 0 = 0 \text{ with a carry of } 1$$

Therefore, $\Sigma = \mathbf{0}$ and $C_{out} = \mathbf{1}$.

(c) The input bits are $A = 1, B = 0$, and $C_{in} = 1$.

$$1 + 0 + 1 = 0 \text{ with a carry of } 1$$

Therefore, $\Sigma = \mathbf{0}$ and $C_{out} = \mathbf{1}$.

Related Problem*

What are the full-adder outputs for $A = 1, B = 1$, and $C_{in} = 1$?

*Answers are at the end of the chapter.

SECTION 6-1 CHECKUP

Answers are at the end of the chapter.

1. Determine the sum (Σ) and the output carry (C_{out}) of a half-adder for each set of input bits:

   (a) 01 (b) 00 (c) 10 (d) 11

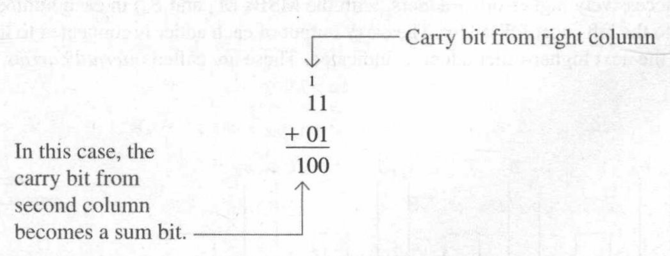
2. A full-adder has $C_{in} = 1$. What are the sum (Σ) and the output carry (C_{out}) when $A = 1$ and $B = 1$?

6-2 Parallel Binary Adders

InfoNote

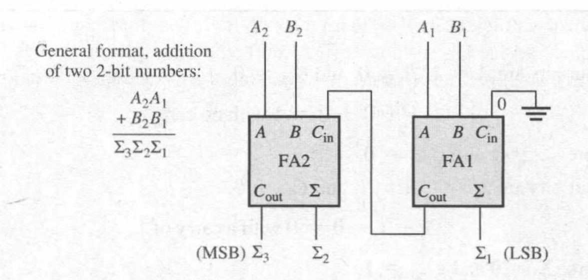
Addition is performed by processors on two numbers at a time, called *operands*. The *source operand* is a number that is to be added to an existing number called the *destination operand*, which is held in an ALU register, such as the accumulator. The sum of the two numbers is then stored back in the accumulator. Addition is performed on integer numbers or floating-point numbers using ADD or FADD instructions respectively.

As you learned in Section 6–1, a single full-adder is capable of adding two 1-bit numbers and an input carry. To add binary numbers with more than one bit, you must use additional full-adders. When one binary number is added to another, each column generates a sum bit and a 1 or 0 carry bit to the next column to the left, as illustrated here with 2-bit numbers.



To add two binary numbers, a full-adder (FA) is required for each bit in the numbers. So for 2-bit numbers, two adders are needed; for 4-bit numbers, four adders are used; and so on. The carry output of each adder is connected to the carry input of the next higher-order adder, as shown in Figure 6–7 for a 2-bit adder. Notice that either a half-adder can be used for the least significant position or the carry input of a full-adder can be made 0 (grounded) because there is no carry input to the least significant bit position.

In Figure 6–7 the least significant bits (LSB) of the two numbers are represented by A_1 and B_1. The next higher-order bits are represented by A_2 and B_2. The three sum bits are Σ_1, Σ_2, and Σ_3. Notice that the output carry from the left-most full-adder becomes the most significant bit (MSB) in the sum, Σ_3.



◀ **FIGURE 6-7**

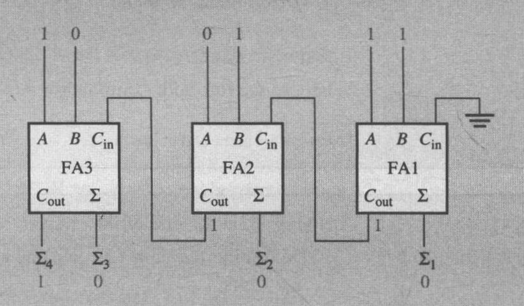
Block diagram of a basic 2-bit parallel adder using two full-adders. Open file F06-07 to verify operation.

MultiSim

EXAMPLE 6-2

Determine the sum generated by the 3-bit parallel adder in Figure 6–8 and show the intermediate carries when the binary numbers 101 and 011 are being added.

▶ **FIGURE 6-8**



Solution

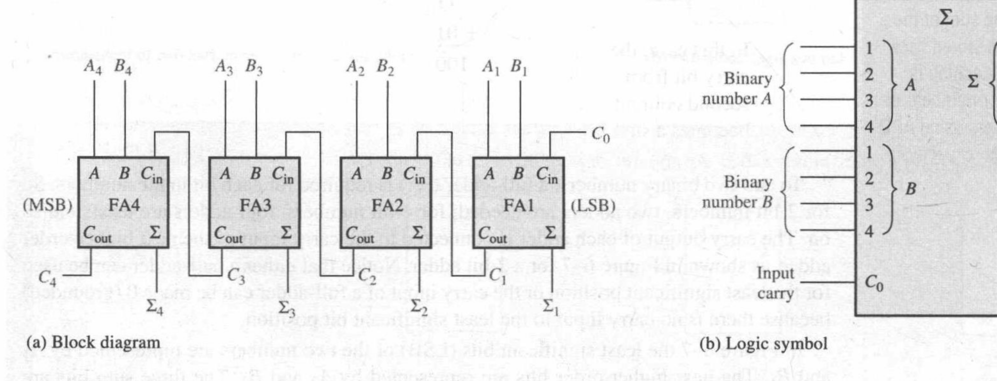
The LSBs of the two numbers are added in the right-most full-adder. The sum bits and the intermediate carries are indicated in blue in Figure 6–8.

Related Problem

What are the sum outputs when 111 and 101 are added by the 3-bit parallel adder?

Four-Bit Parallel Adders

A group of four bits is called a **nibble**. A basic 4-bit parallel adder is implemented with four full-adder stages as shown in Figure 6–9. Again, the LSBs (A_1 and B_1) in each number being added go into the right-most full-adder; the higher-order bits are applied as shown to the successively higher-order adders, with the MSBs (A_4 and B_4) in each number being applied to the left-most full-adder. The carry output of each adder is connected to the carry input of the next higher-order adder as indicated. These are called *internal carries*.



▲ **FIGURE 6-9**

A 4-bit parallel adder.

In keeping with most manufacturers' data sheets, the input labeled C_0 is the input carry to the least significant bit adder; C_4, in the case of four bits, is the output carry of the most significant bit adder; and Σ_1 (LSB) through Σ_4 (MSB) are the sum outputs. The logic symbol is shown in Figure 6–9(b).

In terms of the method used to handle carries in a parallel adder, there are two types: the *ripple carry* adder and the *carry look-ahead* adder. These are discussed in Section 6–3.

Truth Table for a 4-Bit Parallel Adder

Table 6–3 is the truth table for a 4-bit adder. On some data sheets, truth tables may be called *function tables* or *functional truth tables.* The subscript n represents the adder bits and can be 1, 2, 3, or 4 for the 4-bit adder. C_{n-1} is the carry from the previous adder. Carries C_1, C_2, and C_3 are generated internally. C_0 is an external carry input and C_4 is an output. Example 6–3 illustrates how to use Table 6–3.

▶TABLE 6-3

Truth table for each stage of a 4-bit parallel adder.

C_{n-1}	A_n	B_n	Σ_n	C_n
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

EXAMPLE 6-3

Use the 4-bit parallel adder truth table (Table 6–3) to find the sum and output carry for the addition of the following two 4-bit numbers if the input carry (C_{n-1}) is 0:

$$A_4A_3A_2A_1 = 1100 \quad \text{and} \quad B_4B_3B_2B_1 = 1100$$

Solution

For $n = 1$: $A_1 = 0$, $B_1 = 0$, and $C_{n-1} = 0$. From the 1st row of the table,

$$\Sigma_1 = \mathbf{0} \quad \text{and} \quad C_1 = 0$$

For $n = 2$: $A_2 = 0$, $B_2 = 0$, and $C_{n-1} = 0$. From the 1st row of the table,

$$\Sigma_2 = \mathbf{0} \quad \text{and} \quad C_2 = 0$$

For $n = 3$: $A_3 = 1$, $B_3 = 1$, and $C_{n-1} = 0$. From the 4th row of the table,

$$\Sigma_3 = \mathbf{0} \quad \text{and} \quad C_3 = 1$$

For $n = 4$: $A_4 = 1$, $B_4 = 1$, and $C_{n-1} = 1$. From the last row of the table,

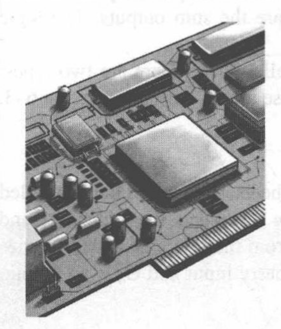
$$\Sigma_4 = \mathbf{1} \quad \text{and} \quad C_4 = \mathbf{1}$$

C_4 becomes the output carry; the sum of 1100 and 1100 is 11000.

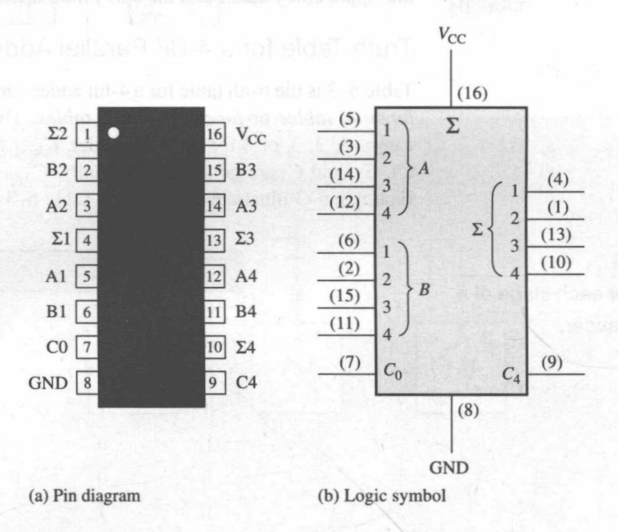
Related Problem

Use the truth table (Table 6–3) to find the result of adding the binary numbers 1011 and 1010.

IMPLEMENTATION: 4-BIT PARALLEL ADDER



Fixed-Function Device The 74HC283 and the 74LS283 are 4-bit parallel adders with identical package pin configurations. The logic symbol and package pin configuration are shown in Figure 6–10. Go to *ti.com* for data sheet information.



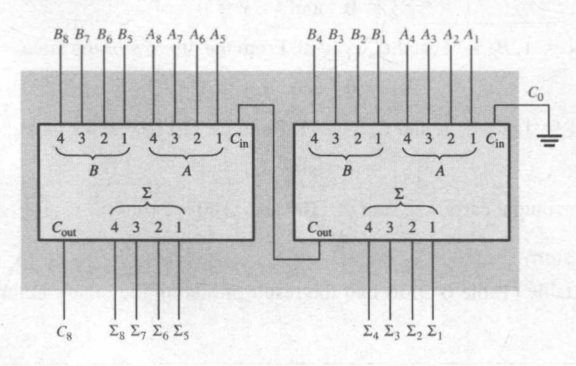
▲FIGURE 6-10

Cascading of two 4-bit adders to form an 8-bit adder.

Adder Expansion

The 4-bit parallel adder can be expanded to handle the addition of two 8-bit numbers by using two 4-bit adders. The carry input of the low-order adder (C_0) is connected to ground because there is no carry into the least significant bit position, and the carry output of the low-order adder is connected to the carry input of the high-order adder, as shown in Figure 6–11. This process is known as **cascading**. Notice that, in this case, the output carry is designated C_8 because it is generated from the eighth bit position. The low-order adder is the one that adds the lower or less significant four bits in the numbers, and the high-order adder is the one that adds the higher or more significant four bits in the 8-bit numbers. Similarly, four 4-bit adders can be cascaded to handle two 16-bit numbers.

Adders can be expanded to handle more bits by cascading.



▲FIGURE 6-11

The 74HC283/74LS283 4-bit parallel adder.

EXAMPLE 6-4

Show how two 74HC283 adders can be connected to form an 8-bit parallel adder. Show output bits for the following 8-bit input numbers:

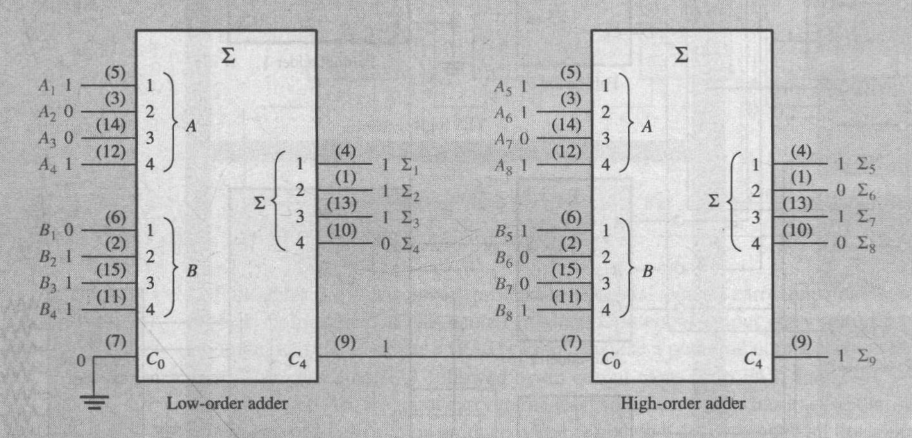
$$A_8A_7A_6A_5A_4A_3A_2A_1 = 10111001 \quad \text{and} \quad B_8B_7B_6B_5B_4B_3B_2B_1 = 10011110$$

Solution

Two 74HC283 4-bit parallel adders are used to implement the 8-bit adder. The only connection between the two 74HC283s is the carry output (pin 9) of the low-order adder to the carry input (pin 7) of the high-order adder, as shown in Figure 6–12. Pin 7 of the low-order adder is grounded (no carry input).

The sum of the two 8-bit numbers is

$$\Sigma_9\Sigma_8\Sigma_7\Sigma_6\Sigma_5\Sigma_4\Sigma_3\Sigma_2\Sigma_1 = 101010111$$



▲ **FIGURE 6-12**

Two 74HC283 adders connected as an 8-bit parallel adder (pin numbers are in parentheses).

Related Problem

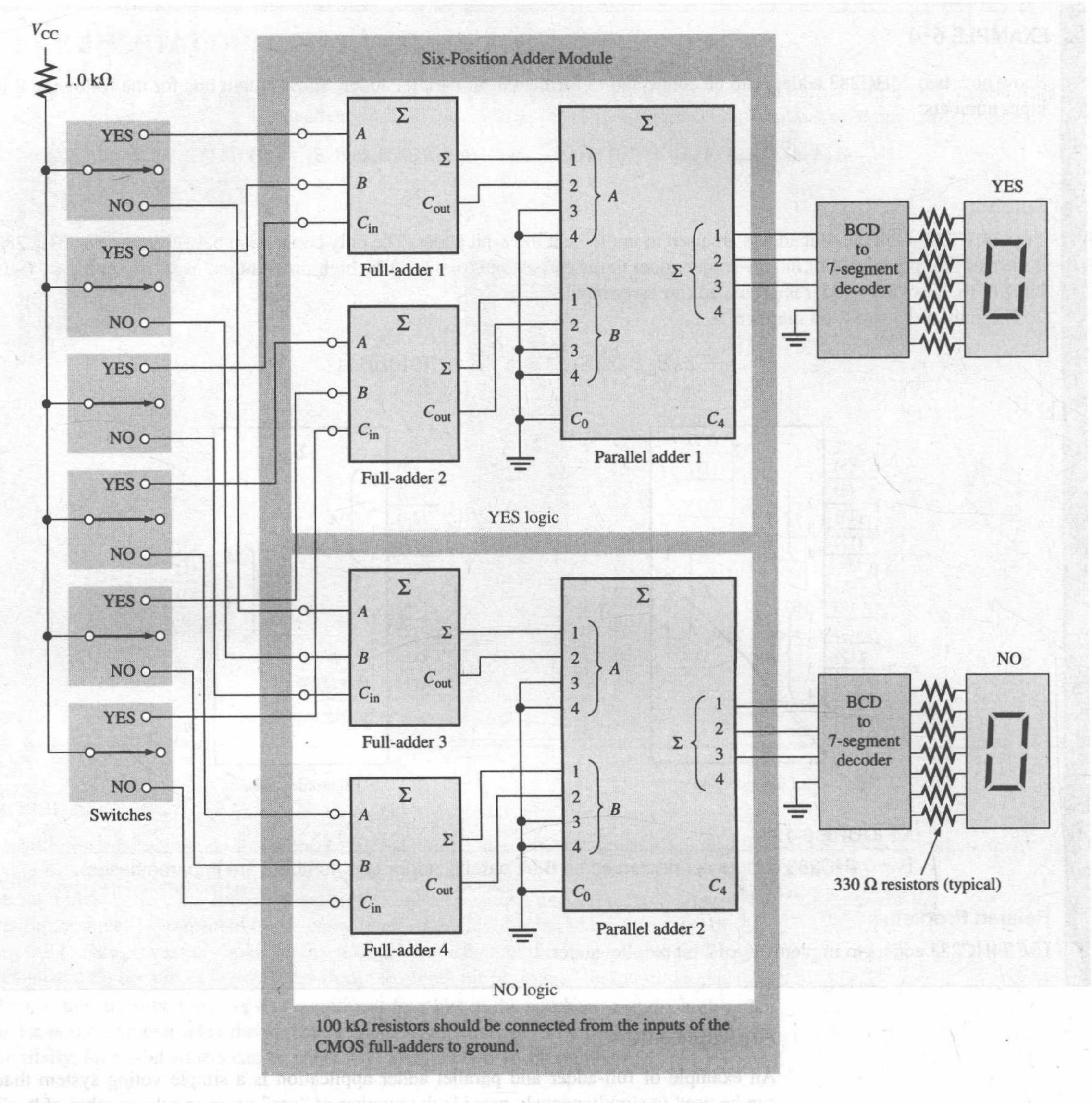
Use 74HC283 adders to implement a 12-bit parallel adder.

An Application

An example of full-adder and parallel adder application is a simple voting system that can be used to simultaneously provide the number of "yes" votes and the number of "no" votes. This type of system can be used where a group of people are assembled and there is a need for immediately determining opinions (for or against), making decisions, or voting on certain issues or other matters.

In its simplest form, the system includes a switch for "yes" or "no" selection at each position in the assembly and a digital display for the number of yes votes and one for the number of no votes. The basic system is shown in Figure 6–13 for a 6-position setup, but it can be expanded to any number of positions with additional 6-position modules and additional parallel adder and display circuits.

In Figure 6–13 each full-adder can produce the sum of up to three votes. The sum and output carry of each full-adder then goes to the two lower-order inputs of a parallel binary adder. The two higher-order inputs of the parallel adder are connected to ground (0) because there is never a case where the binary input exceeds 0011 (decimal 3). For this basic 6-position system, the outputs of the parallel adder go to a BCD-to-7-segment decoder that drives the 7-segment display. As mentioned, additional circuits must be included when the system is expanded.



▲FIGURE 6-13

A voting system using full-adders and parallel binary adders.

The resistors from the inputs of each full-adder to ground assure that each input is LOW when the switch is in the neutral position (CMOS logic is used). When a switch is moved to the "yes" or to the "no" position, a HIGH level (V_{CC}) is applied to the associated full-adder input.

SECTION 6-2 CHECKUP

1. Two 4-bit numbers (1101 and 1011) are applied to a 4-bit parallel adder. The input carry is 1. Determine the sum (Σ) and the output carry.
2. How many 74HC283 adders would be required to add two binary numbers each representing decimal numbers up through 1000_{10}?

6-3 Ripple Carry and Look-Ahead Carry Adders

The Ripple Carry Adder

A **ripple carry** adder is one in which the carry output of each full-adder is connected to the carry input of the next higher-order stage (a stage is one full-adder). The sum and the output carry of any stage cannot be produced until the input carry occurs; this causes a time delay in the addition process, as illustrated in Figure 6–14. The carry propagation delay for each full-adder is the time from the application of the input carry until the output carry occurs, assuming that the A and B inputs are already present.

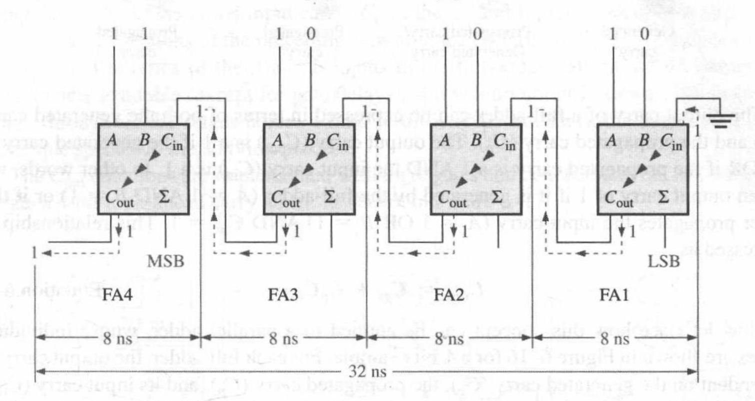


FIGURE 6-14 A 4-bit parallel ripple carry adder showing "worst-case" carry propagation delays.

Full-adder 1 (FA1) cannot produce a potential output carry until an input carry is applied. Full-adder 2 (FA2) cannot produce a potential output carry until FA1 produces an output carry. Full-adder 3 (FA3) cannot produce a potential output carry until an output carry is produced by FA1 followed by an output carry from FA2, and so on. As you can see in Figure 6–14, the input carry to the least significant stage has to ripple through all the adders before a final sum is produced. The cumulative delay through all the adder stages is a "worst-case" addition time. The total delay can vary, depending on the carry bit produced by each full-adder. If two numbers are added such that no carries (0) occur between stages, the addition time is simply the propagation time through a single full-adder from the application of the data bits on the inputs to the occurrence of a sum output; however, worst-case addition time must always be assumed.

The Look-Ahead Carry Adder

The speed with which an addition can be performed is limited by the time required for the carries to propagate, or ripple, through all the stages of a parallel adder. One method of speeding up the addition process by eliminating this ripple carry delay is called **look-ahead carry** addition. The look-ahead carry adder anticipates the output carry of each stage, and based on the inputs, produces the output carry by either carry generation or carry propagation.

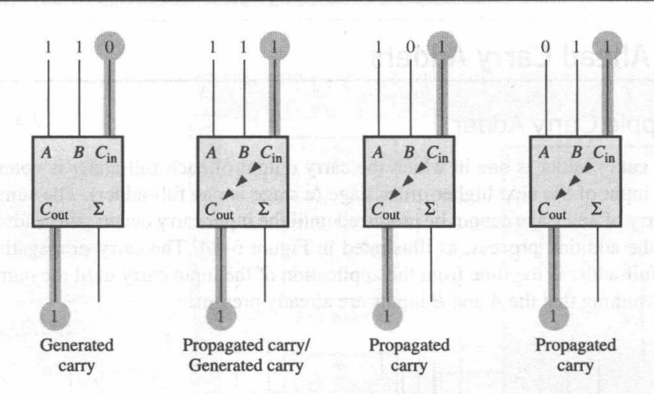
Carry generation occurs when an output carry is produced (generated) internally by the full-adder. A carry is generated only when both input bits are 1s. The generated carry, C_g, is expressed as the AND function of the two input bits, A and B.

$$C_g = AB \qquad \text{Equation 6–5}$$

Carry propagation occurs when the input carry is rippled to become the output carry. An input carry may be propagated by the full-adder when either or both of the input bits are 1s. The propagated carry, C_p, is expressed as the OR function of the input bits.

$$C_p = A + B \qquad \text{Equation 6–6}$$

The conditions for carry generation and carry propagation are illustrated in Figure 6–15. The three arrowheads symbolize ripple (propagation).

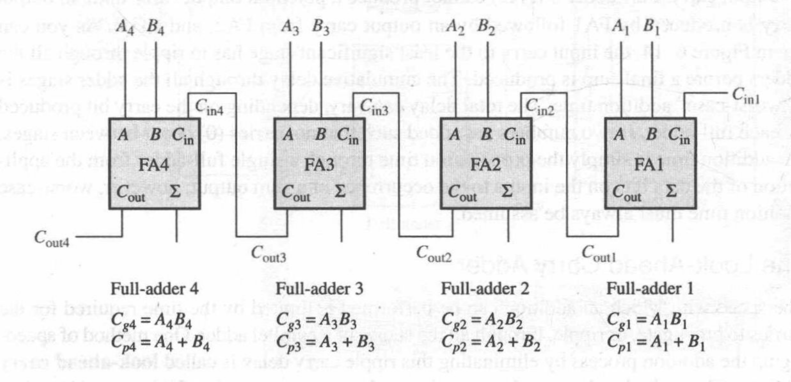


◄ **FIGURE 6-15**
Illustration of conditions for carry generation and carry propagation.

The output carry of a full-adder can be expressed in terms of both the generated carry (C_g) and the propagated carry (C_p). The output carry (C_{out}) is a 1 if the generated carry is a 1 OR if the propagated carry is a 1 AND the input carry (C_{in}) is a 1. In other words, we get an output carry of 1 if it is generated by the full-adder ($A = 1$ AND $B = 1$) or if the adder propagates the input carry ($A = 1$ OR $B = 1$) AND $C_{in} = 1$. This relationship is expressed as

$$C_{out} = C_g + C_pC_{in} \qquad \text{Equation 6–7}$$

Now let's see how this concept can be applied to a parallel adder, whose individual stages are shown in Figure 6–16 for a 4-bit example. For each full-adder, the output carry is dependent on the generated carry (C_g), the propagated carry (C_p), and its input carry (C_{in}). The C_g and C_p functions for each stage are *immediately* available as soon as the input bits A and B and the input carry to the LSB adder are applied because they are dependent only on these bits. The input carry to each stage is the output carry of the previous stage.



◄ **FIGURE 6-16**
Carry generation and carry propagation in terms of the input bits to a 4-bit adder.

Based on this analysis, we can now develop expressions for the output carry, C_{out}, of each full-adder stage for the 4-bit example.

Full-adder 1:

$$C_{out1} = C_{g1} + C_{p1}C_{in1}$$

Full-adder 2:

$$C_{in2} = C_{out1}$$

$$C_{out2} = C_{g2} + C_{p2}C_{in2} = C_{g2} + C_{p2}C_{out1} = C_{g2} + C_{p2}(C_{g1} + C_{p1}C_{in1})$$
$$= C_{g2} + C_{p2}C_{g1} + C_{p2}C_{p1}C_{in1}$$

Full-adder 3:

$$C_{in3} = C_{out2}$$

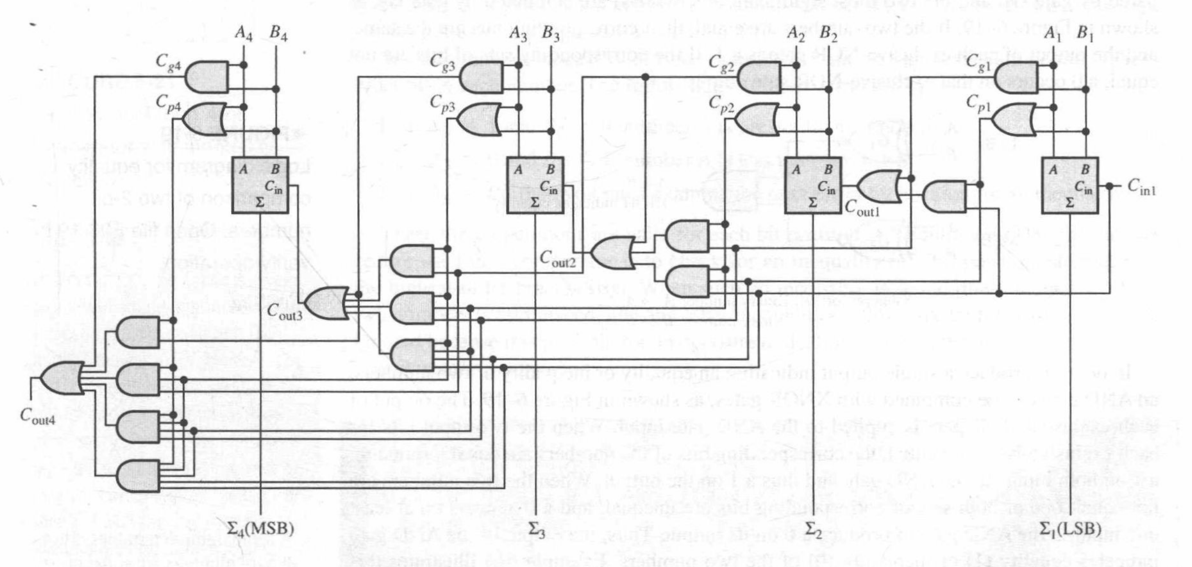
$$C_{out3} = C_{g3} + C_{p3}C_{in3} = C_{g3} + C_{p3}C_{out2} = C_{g3} + C_{p3}(C_{g2} + C_{p2}C_{g1} + C_{p2}C_{p1}C_{in1})$$
$$= C_{g3} + C_{p3}C_{g2} + C_{p3}C_{p2}C_{g1} + C_{p3}C_{p2}C_{p1}C_{in1}$$

Full-adder 4:

$$C_{in4} = C_{out3}$$
$$C_{out4} = C_{g4} + C_{p4}C_{in4} = C_{g4} + C_{p4}C_{out3}$$
$$= C_{g4} + C_{p4}(C_{g3} + C_{p3}C_{g2} + C_{p3}C_{p2}C_{g1} + C_{p3}C_{p2}C_{p1}C_{in1})$$
$$= C_{g4} + C_{p4}C_{g3} + C_{p4}C_{p3}C_{g2} + C_{p4}C_{p3}C_{p2}C_{g1} + C_{p4}C_{p3}C_{p2}C_{p1}C_{in1}$$

Notice that in each of these expressions, the output carry for each full-adder stage is dependent only on the initial input carry (C_{in1}), the C_g and C_p functions of that stage, and the C_g and C_p functions of the preceding stages. Since each of the C_g and C_p functions can be expressed in terms of the A and B inputs to the full-adders, all the output carries are immediately available (except for gate delays), and you do not have to wait for a carry to ripple through all the stages before a final result is achieved. Thus, the look-ahead carry technique speeds up the addition process.

The C_{out} equations are implemented with logic gates and connected to the full-adders to create a 4-bit look-ahead carry adder, as shown in Figure 6–17.



▲**FIGURE 6-17**

Logic diagram for a 4-stage look-ahead carry adder.

Combination Look-Ahead and Ripple Carry Adders

As with most fixed-function IC adders, the 74HC283 4-bit adder that was introduced in Section 6–2 is a look-ahead carry adder. When these adders are cascaded to expand their capability to handle binary numbers with more than four bits, the output carry of one adder is connected to the input carry of the next. This creates a ripple carry condition between the 4-bit adders so that when two or more 74HC283s are cascaded, the resulting adder is actually a combination look-ahead and ripple carry adder. The look-ahead carry operation is internal to each MSI adder and the ripple carry feature comes into play when there is a carry out of one of the adders to the next one.

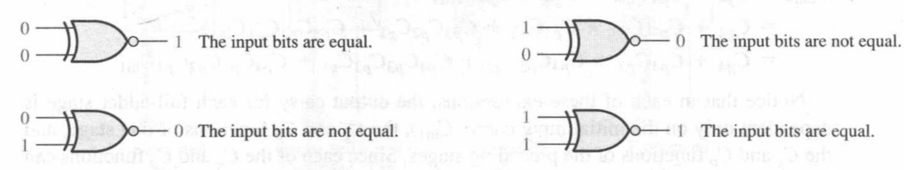
SECTION 6-3 CHECKUP

1. The input bits to a full-adder are $A = 1$ and $B = 0$. Determine C_g and C_p.
2. Determine the output carry of a full-adder when $C_{in} = 1$, $C_g = 0$, and $C_p = 1$.

6-4 Comparators

Equality

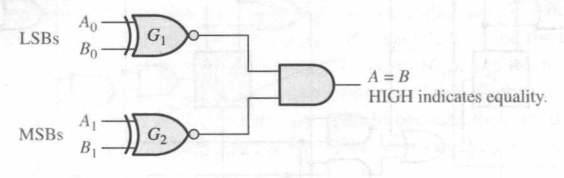
As you learned in Chapter 3, the exclusive-NOR gate can be used as a basic comparator because its output is a 0 if the two input bits are not equal and a 1 if the input bits are equal. Figure 6–18 shows the exclusive-NOR gate as a 2-bit comparator.



▲FIGURE 6-18

Basic comparator operation.

In order to compare binary numbers containing two bits each, an additional exclusive-NOR gate is necessary. The two least significant bits (LSBs) of the two numbers are compared by gate G_1, and the two most significant bits (MSBs) are compared by gate G_2, as shown in Figure 6–19. If the two numbers are equal, their corresponding bits are the same, and the output of each exclusive-NOR gate is a 1. If the corresponding sets of bits are not equal, a 0 occurs on that exclusive-NOR gate output.



◄FIGURE 6-19

Logic diagram for equality comparison of two 2-bit numbers. Open file F06-19 to verify operation.

MultiSim

A comparator determines if two binary numbers are equal or unequal.

In order to produce a single output indicating an equality or inequality of two numbers, an AND gate can be combined with XNOR gates, as shown in Figure 6–19. The output of each exclusive-NOR gate is applied to the AND gate input. When the two input bits for each exclusive-NOR are equal, the corresponding bits of the numbers are equal, producing a 1 on both inputs to the AND gate and thus a 1 on the output. When the two numbers are not equal, one or both sets of corresponding bits are unequal, and a 0 appears on at least one input to the AND gate to produce a 0 on its output. Thus, the output of the AND gate indicates equality (1) or inequality (0) of the two numbers. Example 6–5 illustrates this operation for two specific cases.

EXAMPLE 6-5

Apply each of the following sets of binary numbers to the comparator inputs in Figure 6–20, and determine the output by following the logic levels through the circuit.

(a) 10 and 10 (b) 11 and 10

$A_0 = 0$, $B_0 = 0$ → 1; $A_1 = 1$, $B_1 = 1$ → 1; output: 1 → equal

(a)

$A_0 = 1$, $B_0 = 0$ → 0; $A_1 = 1$, $B_1 = 1$ → 1; output: 0 → not equal

(b)

▲FIGURE 6-20

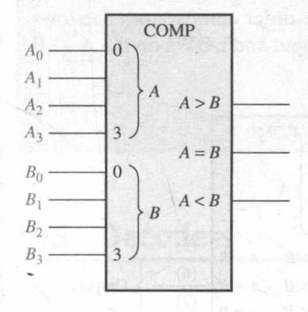
Solution

(a) The output is **1** for inputs 10 and 10, as shown in Figure 6–20(a).

(b) The output is **0** for inputs 11 and 10, as shown in Figure 6–20(b).

Related Problem

Repeat the process for binary inputs of 01 and 10.



▲ **FIGURE 6-21**

Logic symbol for a 4-bit comparator with inequality indication.

As you know from Chapter 3, the basic comparator can be expanded to any number of bits. The AND gate sets the condition that all corresponding bits of the two numbers must be equal if the two numbers themselves are equal.

Inequality

In addition to the equality output, fixed-function comparators can provide additional outputs that indicate which of the two binary numbers being compared is the larger. That is, there is an output that indicates when number *A* is greater than number *B* ($A > B$) and an output that indicates when number *A* is less than number *B* ($A < B$), as shown in the logic symbol for a 4-bit comparator in Figure 6–21.

To determine an inequality of binary numbers *A* and *B*, you first examine the highest-order bit in each number. The following conditions are possible:

1. If $A_3 = 1$ and $B_3 = 0$, number *A* is greater than number *B*.
2. If $A_3 = 0$ and $B_3 = 1$, number *A* is less than number *B*.
3. If $A_3 = B_3$, then you must examine the next lower bit position for an inequality.

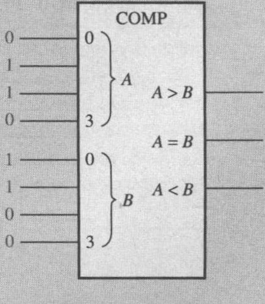
These three operations are valid for each bit position in the numbers. The general procedure used in a comparator is to check for an inequality in a bit position, starting with the highest-order bits (MSBs). When such an inequality is found, the relationship of the two numbers is established, and any other inequalities in lower-order bit positions must be ignored because it is possible for an opposite indication to occur; *the highest-order indication must take precedence.*

InfoNote

In a computer, the *cache* is a very fast intermediate memory between the central processing unit (CPU) and the slower main memory. The CPU requests data by sending out its *address* (unique location) in memory. Part of this address is called a *tag*. The *tag address comparator* compares the tag from the CPU with the tag from the cache directory. If the two agree, the addressed data is already in the cache and is retrieved very quickly. If the tags disagree, the data must be retrieved from the main memory at a much slower rate.

EXAMPLE 6-6

Determine the $A = B$, $A > B$, and $A < B$ outputs for the input numbers shown on the comparator in Figure 6–22.



▲ **FIGURE 6-22**

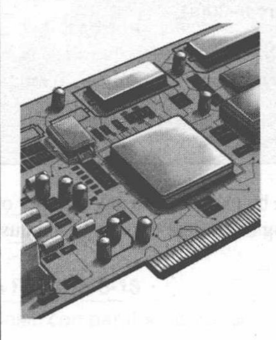
Solution

The number on the *A* inputs is 0110 and the number on the *B* inputs is 0011. The $A > B$ **output is HIGH and the other outputs are LOW.**

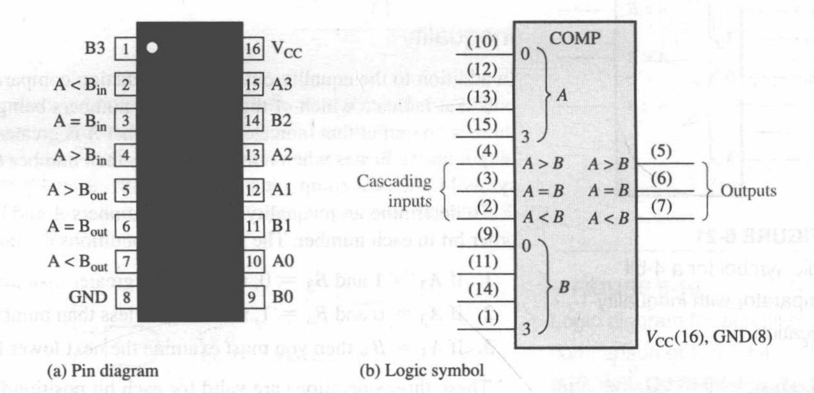
Related Problem

What are the comparator outputs when $A_3A_2A_1A_0 = 1001$ and $B_3B_2B_1B_0 = 1010$?

IMPLEMENTATION: 4-BIT MAGNITUDE COMPARATOR



Fixed-Function Device **The 74HC85/74LS85 pin diagram and logic symbol are shown in Figure 6–23. Notice that this device has all the inputs and outputs of the generalized comparator previously discussed and, in addition, has three cascading inputs: $A < B, A = B, A > B$. These inputs allow several comparators to be cascaded for comparison of any number of bits greater than four. To expand the comparator, the $A < B$, $A = B$, and $A > B$ outputs of the lower-order comparator are connected to the corresponding cascading inputs of the next higher-order comparator. The lowest-order comparator must have a HIGH on the $A = B$ input and LOWs on the $A < B$ and $A > B$ inputs.**



▲ **FIGURE 6-23**
The 74HC85/74LS85 4-bit magnitude comparator.

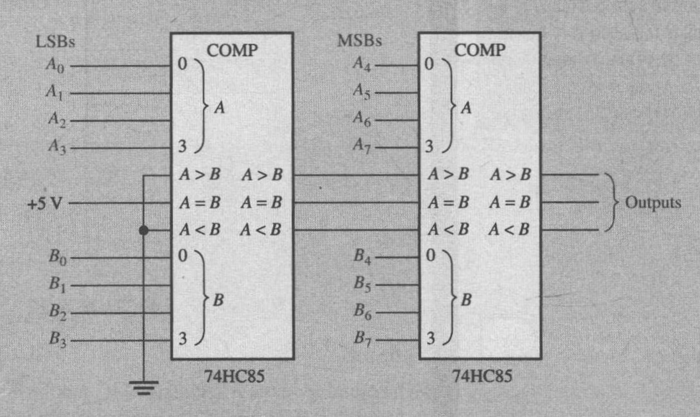
EXAMPLE 6-7

Use 74HC85 comparators to compare the magnitudes of two 8-bit numbers. Show the comparators with proper interconnections.

Solution

Two 74HC85s are required to compare two 8-bit numbers. They are connected as shown in Figure 6–24 in a cascaded arrangement.

► **FIGURE 6-24**
An 8-bit magnitude comparator using two 74HC85s.



Related Problem

Expand the circuit in Figure 6–24 to a 16-bit comparator.

Most CMOS devices contain protection circuitry to guard against damage from high static voltages or electric fields. However, precautions must be taken to avoid applications of any voltages higher than maximum rated voltages. For proper operation, input and output voltages should be between ground and V_{CC}. Also, remember that unused inputs must always be connected to an appropriate logic level (ground or V_{CC}). Unused outputs may be left open.

SECTION 6-4 CHECKUP

1. The binary numbers $A = 1011$ and $B = 1010$ are applied to the inputs of a 74HC85. Determine the outputs.
2. The binary numbers $A = 11001011$ and $B = 11010100$ are applied to the 8-bit comparator in Figure 6–24. Determine the states of the outputs on each comparator.

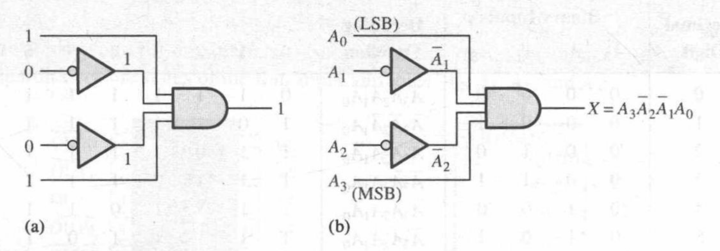
6-5 Decoders

The Basic Binary Decoder

Suppose you need to determine when a binary 1001 occurs on the inputs of a digital circuit. An AND gate can be used as the basic decoding element because it produces a HIGH output only when all of its inputs are HIGH. Therefore, you must make sure that all of the inputs to the AND gate are HIGH when the binary number 1001 occurs; this can be done by inverting the two middle bits (the 0s), as shown in Figure 6–25.

InfoNote

An *instruction* tells the processor what operation to perform. Instructions are in machine code (1s and 0s) and, in order for the processor to carry out an instruction, the instruction must be decoded. Instruction decoding is one of the steps in instruction *pipelining*, which are as follows: Instruction is read from the memory (instruction fetch), instruction is decoded, operand(s) is (are) read from memory (operand fetch), instruction is executed, and result is written back to memory. Basically, pipelining allows the next instruction to begin processing before the current one is completed.



▲ **FIGURE 6-25**

Decoding logic for the binary code 1001 with an active-HIGH output.

The logic equation for the decoder of Figure 6–25(a) is developed as illustrated in Figure 6–25(b). You should verify that the output is 0 except when $A_0 = 1$, $A_1 = 0$, $A_2 = 0$, and $A_3 = 1$ are applied to the inputs. A_0 is the LSB and A_3 is the MSB. *In the representation of a binary number or other weighted code in this book, the LSB is the right-most bit in a horizontal arrangement and the topmost bit in a vertical arrangement, unless specified otherwise.*

If a NAND gate is used in place of the AND gate in Figure 6–25, a LOW output will indicate the presence of the proper binary code, which is 1001 in this case.

EXAMPLE 6-8

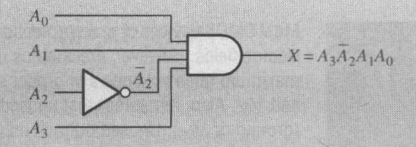
Determine the logic required to decode the binary number 1011 by producing a HIGH level on the output.

Solution

The decoding function can be formed by complementing only the variables that appear as 0 in the desired binary number, as follows:

$$X = A_3\overline{A}_2A_1A_0 \quad (1011)$$

This function can be implemented by connecting the true (uncomplemented) variables A_0, A_1, and A_3 directly to the inputs of an AND gate, and inverting the variable A_2 before applying it to the AND gate input. The decoding logic is shown in Figure 6–26.



▶ **FIGURE 6-26**
Decoding logic for producing a HIGH output when 1011 is on the inputs.

Related Problem

Develop the logic required to detect the binary code 10010 and produce an active-LOW output.

The 4-Bit Decoder

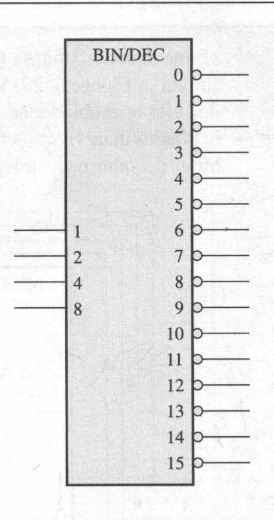
In order to decode all possible combinations of four bits, sixteen decoding gates are required ($2^4 = 16$). This type of decoder is commonly called either a *4-line-to-16-line decoder* because there are four inputs and sixteen outputs or a *1-of-16 decoder* because for any given code on the inputs, one of the sixteen outputs is activated. A list of the sixteen binary codes and their corresponding decoding functions is given in Table 6–4.

▼ **TABLE 6-4**
Decoding functions and truth table for a 4-line-to-16-line (1-of-16) decoder with active-LOW outputs.

Decimal Digit	Binary Inputs A_3	A_2	A_1	A_0	Decoding Function	Outputs 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	$\overline{A_3}\overline{A_2}\overline{A_1}\overline{A_0}$	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	1	$\overline{A_3}\overline{A_2}\overline{A_1}A_0$	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	0	0	1	0	$\overline{A_3}\overline{A_2}A_1\overline{A_0}$	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
3	0	0	1	1	$\overline{A_3}\overline{A_2}A_1A_0$	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
4	0	1	0	0	$\overline{A_3}A_2\overline{A_1}\overline{A_0}$	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
5	0	1	0	1	$\overline{A_3}A_2\overline{A_1}A_0$	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
6	0	1	1	0	$\overline{A_3}A_2A_1\overline{A_0}$	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
7	0	1	1	1	$\overline{A_3}A_2A_1A_0$	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
8	1	0	0	0	$A_3\overline{A_2}\overline{A_1}\overline{A_0}$	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
9	1	0	0	1	$A_3\overline{A_2}\overline{A_1}A_0$	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
10	1	0	1	0	$A_3\overline{A_2}A_1\overline{A_0}$	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
11	1	0	1	1	$A_3\overline{A_2}A_1A_0$	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
12	1	1	0	0	$A_3A_2\overline{A_1}\overline{A_0}$	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
13	1	1	0	1	$A_3A_2\overline{A_1}A_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
14	1	1	1	0	$A_3A_2A_1\overline{A_0}$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
15	1	1	1	1	$A_3A_2A_1A_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

If an active-LOW output is required for each decoded number, the entire decoder can be implemented with NAND gates and inverters. In order to decode each of the sixteen binary codes, sixteen NAND gates are required (AND gates can be used to produce active-HIGH outputs).

A logic symbol for a 4-line-to-16-line (1-of-16) decoder with active-LOW outputs is shown in Figure 6–27. The BIN/DEC label indicates that a binary input makes the corresponding decimal output active. The input labels 8, 4, 2, and 1 represent the binary weights of the input bits ($2^3 2^2 2^1 2^0$).

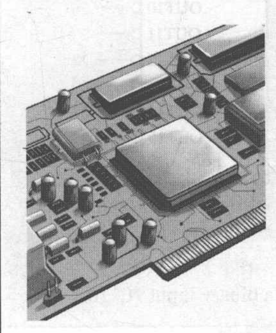


▶ **FIGURE 6-27**

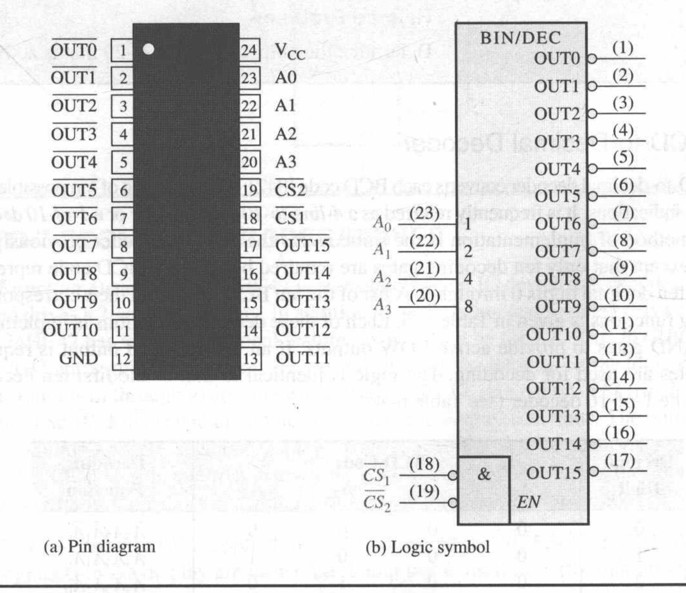
Logic symbol for a 4-line-to-16-line (1-of-16) ecoder. Open file F06-28 to verify operation.

MultiSim

IMPLEMENTATION: 1-OF-16 DECODER



Fixed-Function Device The 74HC154 is a good example of a fixed-function IC decoder. The logic symbol is shown in Figure 6–28. There is an enable function (*EN*) provided on this device, which is implemented with a NOR gate used as a negative-AND. A LOW level on each chip select input, $\overline{CS}_1$ and $\overline{CS}_2$, is required in order to make the enable gate output (*EN*) HIGH. The enable gate output is connected to an input of *each* NAND gate in the decoder, so it must be HIGH for the NAND gates to be enabled. If the enable gate is not activated by a LOW on both inputs, then all sixteen decoder outputs (*OUT*) will be HIGH regardless of the states of the four input variables, A_0, A_1, A_2, and A_3.



▶ **FIGURE 6-28**

The 74HC154 1-of-16 decoder.

EXAMPLE 6-9

A certain application requires that a 5-bit number be decoded. Use 74HC154 decoders to implement the logic. The binary number is represented by the format $A_4A_3A_2A_1A_0$.

Solution

Since the 74HC154 can handle only four bits, two decoders must be used to form a 5-bit expansion. The fifth bit, A_4, is connected to the chip select inputs, $\overline{CS}_1$ and $\overline{CS}_2$,

of one decoder, and $\overline{A_4}$ is connected to the $\overline{CS_1}$ and $\overline{CS_2}$ inputs of the other decoder, as shown in Figure 6–29. When the decimal number is 15 or less, $A_4 = 0$, the low-order decoder is enabled, and the high-order decoder is disabled. When the decimal number is greater than 15, $A_4 = 1$ so $\overline{A_4} = 0$, the high-order decoder is enabled, and the low-order decoder is disabled.

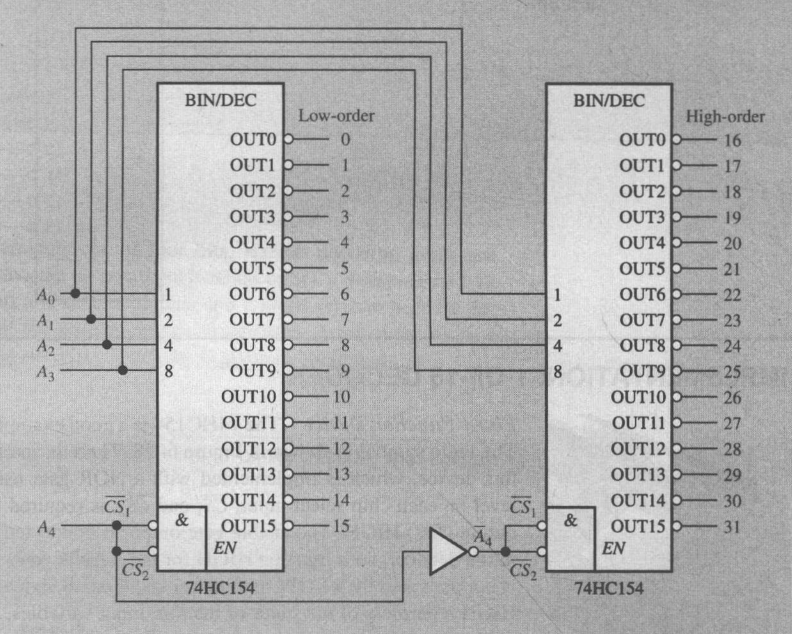


FIGURE 6-29
A 5-bit decoder using 74HC154s.

Related Problem

Determine the output in Figure 6–29 that is activated for the binary input 10110.

The BCD-to-Decimal Decoder

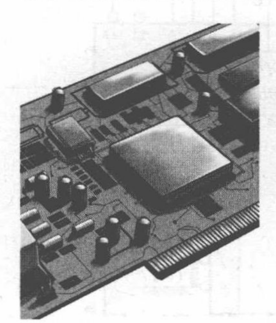
The BCD-to-decimal decoder converts each BCD code (8421 code) into one of ten possible decimal digit indications. It is frequently referred as a *4-line-to-10-line decoder* or a *1-of-10 decoder.*

The method of implementation is the same as for the 1-of-16 decoder previously discussed, except that only ten decoding gates are required because the BCD code represents only the ten decimal digits 0 through 9. A list of the ten BCD codes and their corresponding decoding functions is given in Table 6–5. Each of these decoding functions is implemented with NAND gates to provide active-LOW outputs. If an active-HIGH output is required, AND gates are used for decoding. The logic is identical to that of the first ten decoding gates in the 1-of-16 decoder (see Table 6–4).

TABLE 6-5
BCD decoding functions.

Decimal Digit	BCD Code A_3	A_2	A_1	A_0	Decoding Function
0	0	0	0	0	$\overline{A}_3\overline{A}_2\overline{A}_1\overline{A}_0$
1	0	0	0	1	$\overline{A}_3\overline{A}_2\overline{A}_1A_0$
2	0	0	1	0	$\overline{A}_3\overline{A}_2A_1\overline{A}_0$
3	0	0	1	1	$\overline{A}_3\overline{A}_2A_1A_0$
4	0	1	0	0	$\overline{A}_3A_2\overline{A}_1\overline{A}_0$
5	0	1	0	1	$\overline{A}_3A_2\overline{A}_1A_0$
6	0	1	1	0	$\overline{A}_3A_2A_1\overline{A}_0$
7	0	1	1	1	$\overline{A}_3A_2A_1A_0$
8	1	0	0	0	$A_3\overline{A}_2\overline{A}_1\overline{A}_0$
9	1	0	0	1	$A_3\overline{A}_2\overline{A}_1A_0$

IMPLEMENTATION: BCD-TO-DECIMAL DECODER



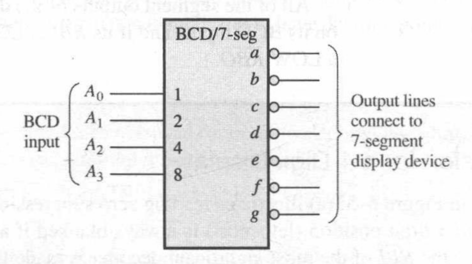
Fixed-Function Device The 74HC42 is a fixed-function IC decoder with four BCD inputs and ten active-LOW decimal outputs. The logic symbol is shown in Figure 6–30.

▶ **FIGURE 6-30**
The 74HC42 BCD-to-decimal decoder.

BCD/DEC
A_0 (15) 1
A_1 (14) 2
A_2 (13) 4
A_3 (12) 8
OUT0 (1)
OUT1 (2)
OUT2 (3)
OUT3 (4)
OUT4 (5)
OUT5 (6)
OUT6 (7)
OUT7 (9)
OUT8 (10)
OUT9 (11)
74HC42

The BCD-to-7-Segment Decoder

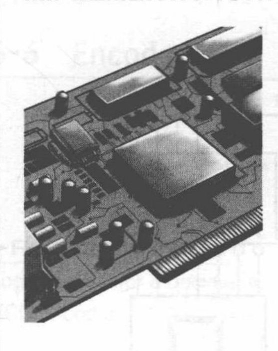
The BCD-to-7-segment decoder accepts the BCD code on its inputs and provides outputs to drive 7-segment display devices to produce a decimal readout. The logic diagram for a basic 7-segment decoder is shown in Figure 6–31.



▶ **FIGURE 6-31**
Logic symbol for a BCD-to-7-segment decoder/driver with active-LOW outputs. Open file F06-33 to verify operation.

MultiSim

IMPLEMENTATION: BCD-TO-7-SEGMENT DECODER/DRIVER



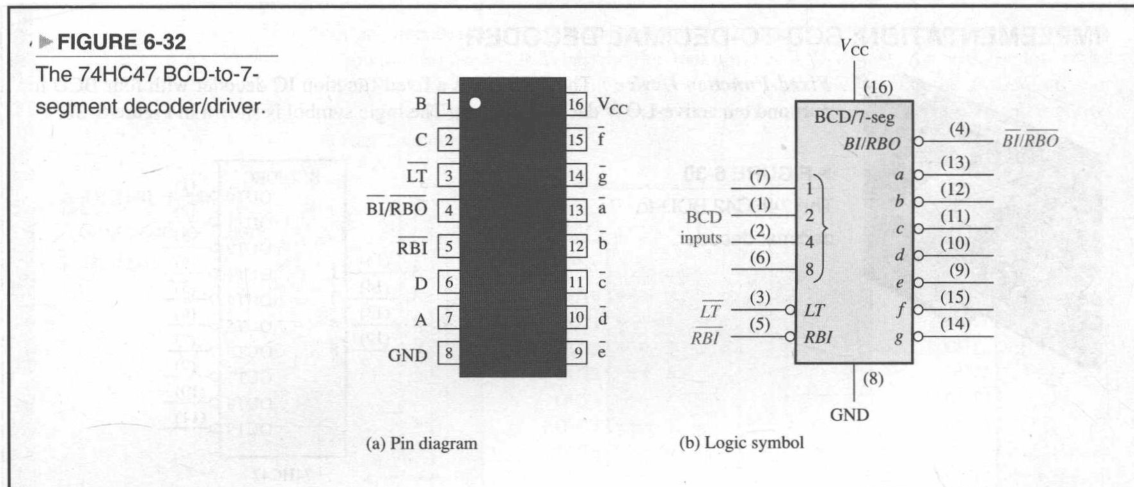
Fixed-Function Device The 74HC47 is an example of an IC device that decodes a BCD input and drives a 7-segment display. In addition to its decoding and segment drive capability, the 74HC47 has several additional features as indicated by the $\overline{LT}$, $\overline{RBI}$, $\overline{BI}/\overline{RBO}$ functions in the logic symbol of Figure 6–32. As indicated by the bubbles on the logic symbol, all of the outputs (*a* through *g*) are active-LOW as are the $\overline{LT}$ (lamp test), $\overline{RBI}$ (ripple blanking input), and $\overline{BI}/\overline{RBO}$ (blanking input/ripple blanking output) functions. The outputs can drive a common-anode 7-segment display directly. Recall that 7-segment displays were discussed in Chapter 4. In addition to decoding a BCD input and producing the appropriate 7-segment outputs, the 74HC47 has lamp test and zero suppression capability.

Lamp Test When a LOW is applied to the $\overline{LT}$ input and the $\overline{BI}/\overline{RBO}$ is HIGH, all of the seven segments in the display are turned on. Lamp test is used to verify that no segments are burned out.

Zero Suppression **Zero suppression** is a feature used for multidigit displays to blank out unnecessary zeros. For example, in a 6-digit display the number 6.4 may be displayed as 006.400 if the zeros are not blanked out. Blanking the zeros at the front of a number is called *leading zero suppression* and blanking the zeros at the back of the number is called *trailing zero suppression.* Keep in mind that only nonessential zeros are blanked. With zero suppression, the number 030.080 will be displayed as 30.08 (the essential zeros remain).

FIGURE 6-32

The 74HC47 BCD-to-7-segment decoder/driver.



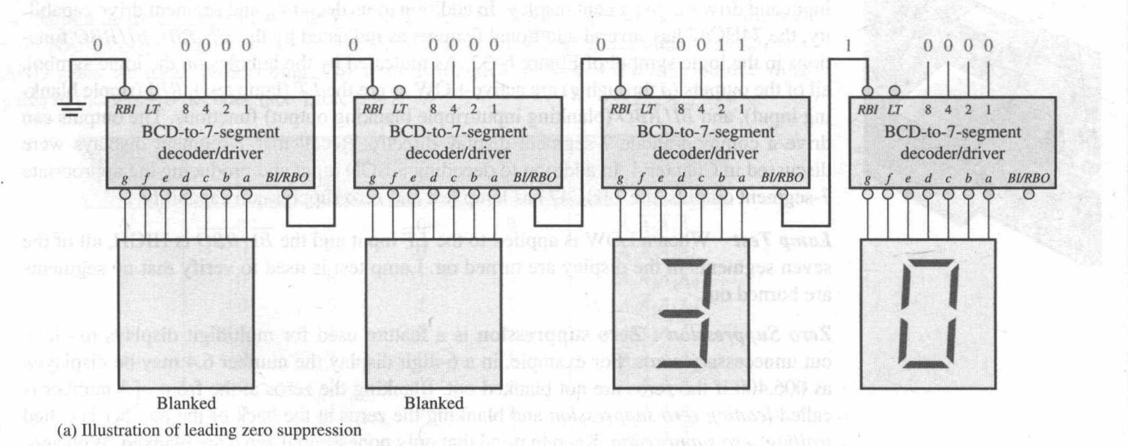
Zero suppression in the 74HC47 is accomplished using the $\overline{RBI}$ and $\overline{BI}/\overline{RBO}$ functions. $\overline{RBI}$ is the ripple blanking input and $\overline{RBO}$ is the ripple blanking output on the 74HC47; these are used for zero suppression. $\overline{BI}$ is the blanking input that shares the same pin with $\overline{RBO}$; in other words, the $\overline{BI}/\overline{RBO}$ pin can be used as an input or an output. When used as a $\overline{BI}$ (blanking input), all segment outputs are HIGH (nonactive) when $\overline{BI}$ is LOW, which overrides all other inputs. The $\overline{BI}$ function is not part of the zero suppression capability of the device.

All of the segment outputs of the decoder are nonactive (HIGH) if a zero code (0000) is on its BCD inputs and if its $\overline{RBI}$ is LOW. This causes the display to be blank and produces a LOW $\overline{RBO}$.

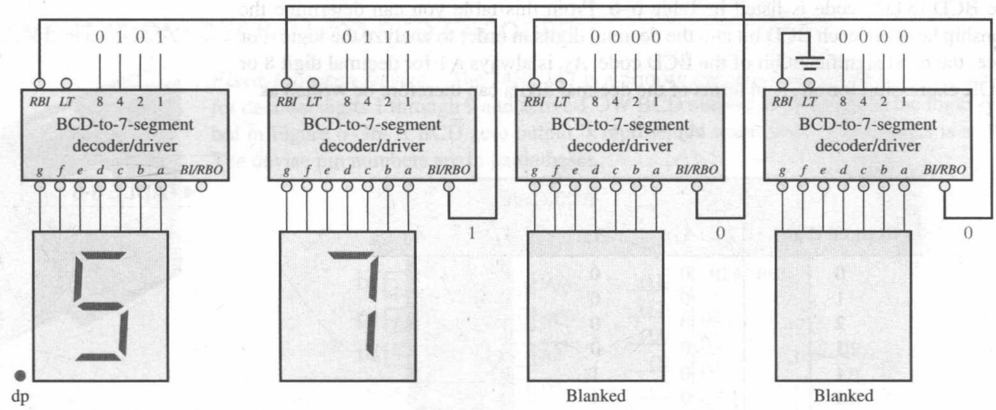
Zero Suppression for a 4-Digit Display

The logic diagram in Figure 6–33(a) illustrates leading zero suppression for a whole number. The highest-order digit position (left-most) is always blanked if a zero code is on its BCD inputs because the $\overline{RBI}$ of the most-significant decoder is made LOW by connecting it to ground. The $\overline{RBO}$ of each decoder is connected to the $\overline{RBI}$ of the next lowest-order decoder so that all zeros to the left of the first nonzero digit are blanked. For example, in part (a) of the figure the two highest-order digits are zeros and therefore are blanked. The remaining two digits, 3 and 0 are displayed.

Zero suppression results in leading or trailing zeros in a number not showing on a display.



(a) Illustration of leading zero suppression



(b) Illustration of trailing zero suppression

▲ **FIGURE 6-33**

Examples of zero suppression using a BCD-to-7-segment decoder/driver.

The logic diagram in Figure 6–33(b) illustrates trailing zero suppression for a fractional number. The lowest-order digit (right-most) is always blanked if a zero code is on its BCD inputs because the $\overline{RBI}$ is connected to ground. The $\overline{RBO}$ of each decoder is connected to the $\overline{RBI}$ of the next highest-order decoder so that all zeros to the right of the first nonzero digit are blanked. In part (b) of the figure, the two lowest-order digits are zeros and therefore are blanked. The remaining two digits, 5 and 7 are displayed. To combine both leading and trailing zero suppression in one display and to have decimal point capability, additional logic is required.

SECTION 6-5 CHECKUP

1. A 3-line-to-8-line decoder can be used for octal-to-decimal decoding. When a binary 101 is on the inputs, which output line is activated?
2. How many 74HC154 1-of-16 decoders are necessary to decode a 6-bit binary number?
3. Would you select a decoder/driver with active-HIGH or active-LOW outputs to drive a common-cathode 7-segment LED display?

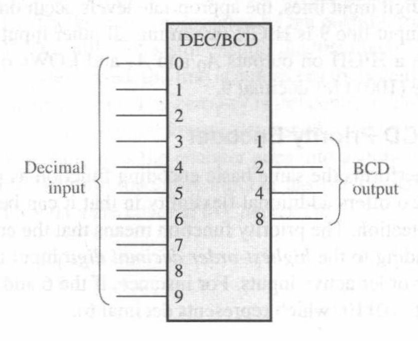
6-6 Encoders

The Decimal-to-BCD Encoder

This type of encoder has ten inputs—one for each decimal digit—and four outputs corresponding to the BCD code, as shown in Figure 6–34. This is a basic 10-line-to-4-line encoder.

► **FIGURE 6-34**

Logic symbol for a decimal-to-BCD encoder.



The BCD (8421) code is listed in Table 6–6. From this table you can determine the relationship between each BCD bit and the decimal digits in order to analyze the logic. For instance, the most significant bit of the BCD code, A_3, is always a 1 for decimal digit 8 or 9. An OR expression for bit A_3 in terms of the decimal digits can therefore be written as

$$A_3 = 8 + 9$$

◀ **TABLE 6-6**

Decimal Digit	BCD Code			
	A_3	A_2	A_1	A_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Bit A_2 is always a 1 for decimal digit 4, 5, 6 or 7 and can be expressed as an OR function as follows:

$$A_2 = 4 + 5 + 6 + 7$$

Bit A_1 is always a 1 for decimal digit 2, 3, 6, or 7 and can be expressed as

$$A_1 = 2 + 3 + 6 + 7$$

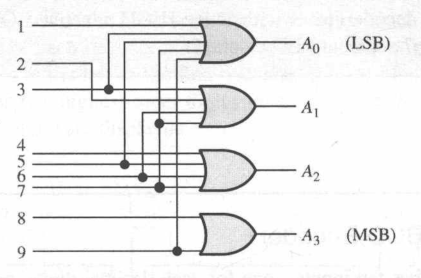
Finally, A_0 is always a 1 for decimal digit 1, 3, 5, 7, or 9. The expression for A_0 is

$$A_0 = 1 + 3 + 5 + 7 + 9$$

Now let's implement the logic circuitry required for encoding each decimal digit to a BCD code by using the logic expressions just developed. It is simply a matter of ORing the appropriate decimal digit input lines to form each BCD output. The basic encoder logic resulting from these expressions is shown in Figure 6–35.

InfoNote

An *assembler* can be thought of as a software encoder because it interprets the mnemonic instructions with which a program is written and carries out the applicable encoding to convert each mnemonic to a machine code instruction (series of 1s and 0s) that the processor can understand. Examples of mnemonic instructions for a processor are ADD, MOV (move data), MUL (multiply), XOR, JMP (jump), and OUT (output to a port).



◀ **FIGURE 6-35**

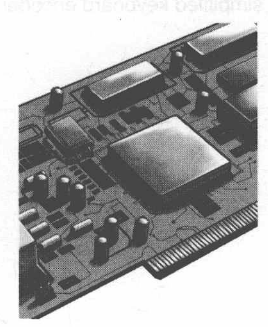
Basic logic diagram of a decimal-to-BCD encoder. A 0-digit input is not needed because the BCD outputs are all LOW when there are no HIGH inputs.

The basic operation of the circuit in Figure 6–35 is as follows: When a HIGH appears on *one* of the decimal digit input lines, the appropriate levels occur on the four BCD output lines. For instance, if input line 9 is HIGH (assuming all other input lines are LOW), this condition will produce a HIGH on outputs A_0 and A_3 and LOWs on outputs A_1 and A_2, which is the BCD code (1001) for decimal 9.

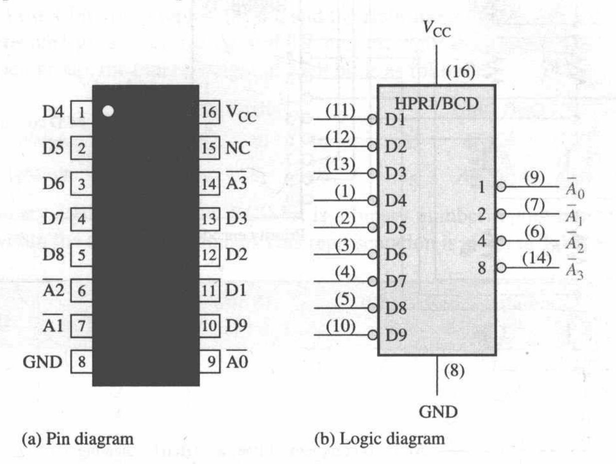
The Decimal-to-BCD Priority Encoder

This type of encoder performs the same basic encoding function as previously discussed. A **priority encoder** also offers additional flexibility in that it can be used in applications that require priority detection. The priority function means that the encoder will produce a BCD output corresponding to the *highest-order decimal digit* input that is active and will ignore any other lower-order active inputs. For instance, if the 6 and the 3 inputs are both active, the BCD output is 0110 (which represents decimal 6).

IMPLEMENTATION: DECIMAL-TO-BCD ENCODER



Fixed-Function Device The 74HC147 is a priority encoder with active-LOW inputs (0) for decimal digits 1 through 9 and active-LOW BCD outputs as indicated in the logic symbol in Figure 6–36. A BCD zero output is represented when none of the inputs is active. The device pin numbers are in parentheses.



▲**FIGURE 6-36**

The 74HC147 decimal-to-BCD encoder (HPRI means highest value input has priority.

EXAMPLE 6-10

If LOW levels appear on pins, 1, 4, and 13 of the 74HC147 shown in Figure 6–36, indicate the state of the four outputs. All other inputs are HIGH.

Solution

Pin 4 is the highest-order decimal digit input having a LOW level and represents decimal 7. Therefore, the output levels indicate the BCD code for decimal 7 where $\overline{A}_0$ is the LSB and $\overline{A}_3$ is the MSB. Output $\overline{A}_0$ is LOW, $\overline{A}_1$ is LOW, $\overline{A}_2$ is LOW, and $\overline{A}_3$ is HIGH.

Related Problem

What are the outputs of the 74HC147 if all its inputs are LOW? If all its inputs are HIGH?

An Application

The ten decimal digits on a numeric keypad must be encoded for processing by the logic circuitry. In this example, when one of the keys is pressed, the decimal digit is encoded to the corresponding BCD code. Figure 6–37 shows a simple keyboard encoder arrangement using a priority encoder. The keys are represented by ten push-button switches, each with a **pull-up resistor** to $+V$. The pull-up resistor ensures that the line is HIGH when a key is not depressed. When a key is depressed, the line is connected to ground, and a LOW is applied to the corresponding encoder input. The zero key is not connected because the BCD output represents zero when none of the other keys is depressed.

The BCD complement output of the encoder goes into a storage device, and each successive BCD code is stored until the entire number has been entered. Methods of storing BCD numbers and binary data are covered in Chapter 10.

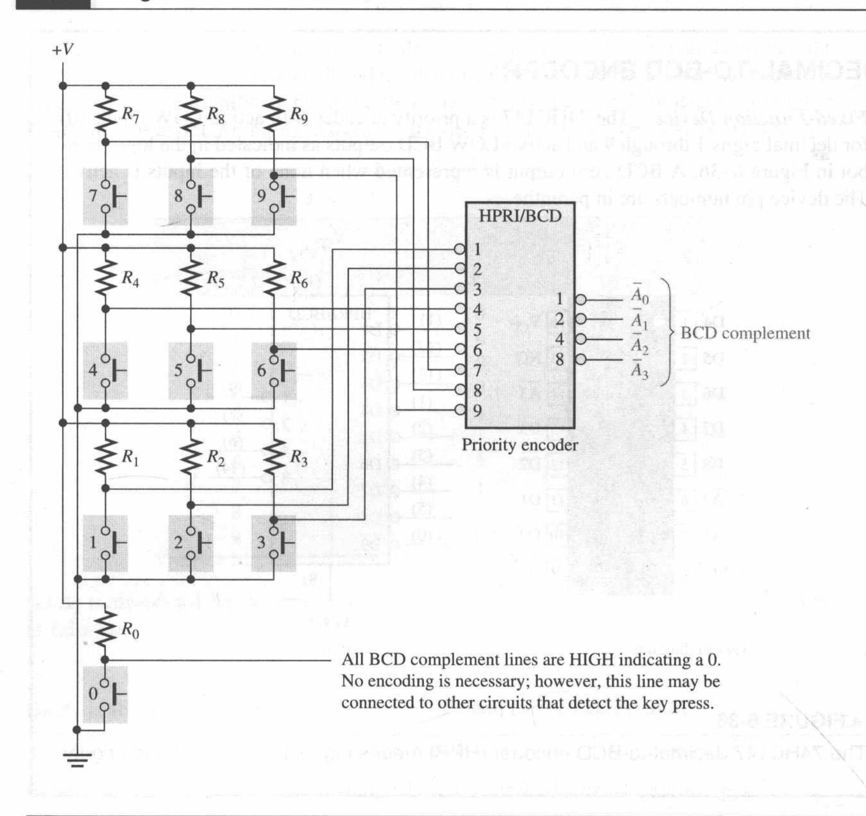


FIGURE 6-37
A simplified keyboard encoder.

SECTION 6-6 CHECKUP

1. Suppose the HIGH levels are applied to the 2 input and the 9 input of the circuit in Figure 6–35.
   (a) What are the states of the output lines?
   (b) Does this represent a valid BCD code?
   (c) What is the restriction on the encoder logic in Figure 6–35?
2. (a) What is the $\overline{A}_3\overline{A}_2\overline{A}_1\overline{A}_0$ output when LOWs are applied to pins 1 and 5 of the 74HC147 in Figure 6–36?
   (b) What does this output represent?

6-7 Code Converters

BCD-to-Binary Conversion

One method of BCD-to-binary code conversion uses adder circuits. The basic conversion process is as follows:

1. The value, or weight, of each bit in the BCD number is represented by a binary number.
2. All of the binary representations of the weights of bits that are 1s in the BCD number are added.
3. The result of this addition is the binary equivalent of the BCD number.

A more concise statement of this operation is

The binary numbers representing the weights of the BCD bits are summed to produce the total binary number.

Let's examine an 8-bit BCD code (one that represents a 2-digit decimal number) to understand the relationship between BCD and binary. For instance, you already know that the decimal number 87 can be expressed in BCD as

$$\underbrace{1000}_{8}\ \underbrace{0111}_{7}$$

The left-most 4-bit group represents 80, and the right-most 4-bit group represents 7. That is, the left-most group has a weight of 10, and the right-most group has a weight of 1. Within each group, the binary weight of each bit is as follows:

	Tens Digit				**Units Digit**			
Weight:	80	40	20	10	8	4	2	1
Bit designation:	B_3	B_2	B_1	B_0	A_3	A_2	A_1	A_0

The binary equivalent of each BCD bit is a binary number representing the weight of that bit within the total BCD number. This representation is given in Table 6–7.

TABLE 6-7

Binary representations of BCD bit weights.

		Binary Representation						
BCD Bit	**BCD Weight**	**(MSB) 64**	**32**	**16**	**8**	**4**	**2**	**(LSB) 1**
A_0	1	0	0	0	0	0	0	1
A_1	2	0	0	0	0	0	1	0
A_2	4	0	0	0	0	1	0	0
A_3	8	0	0	0	1	0	0	0
B_0	10	0	0	0	1	0	1	0
B_1	20	0	0	1	0	1	0	0
B_2	40	0	1	0	1	0	0	0
B_3	80	1	0	1	0	0	0	0

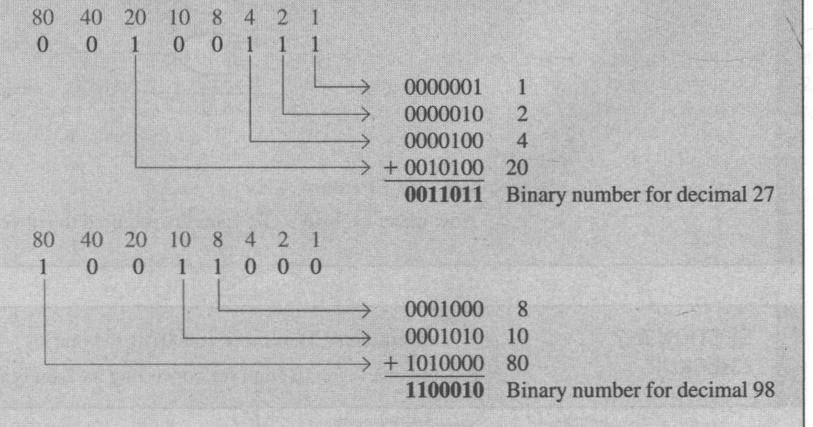
If the binary representations for the weights of all the 1s in the BCD number are added, the result is the binary number that corresponds to the BCD number. Example 6–11 illustrates this.

EXAMPLE 6-11

Convert the BCD numbers 00100111 (decimal 27) and 10011000 (decimal 98) to binary.

Solution

Write the binary representations of the weights of all 1s appearing in the numbers, and then add them together.



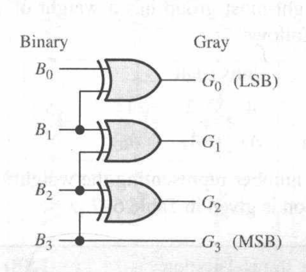
Related Problem

Show the process of converting 01000001 in BCD to binary.

MultiSim Open file EX06-12 and run the simulation to observe the operation of a BCD-to-binary logic circuit.

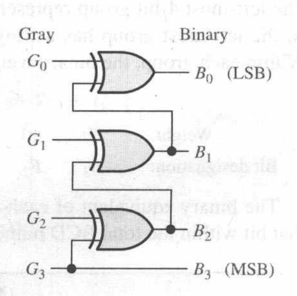
Binary-to-Gray and Gray-to-Binary Conversion

The basic process for Gray-binary conversions was covered in Chapter 2. Exclusive-OR gates can be used for these conversions. Programmable logic devices (PLDs) can also be programmed for these code conversions. Figure 6–38 shows a 4-bit binary-to-Gray code converter, and Figure 6–39 illustrates a 4-bit Gray-to-binary converter.



▲ **FIGURE 6-38**

MultiSim Four-bit binary-to-Gray conversion logic. Open file F06-40 to verify operation.



▲ **FIGURE 6-39**

MultiSim Four-bit Gray-tobinary conversion logic. Open file F06-41 to verify operation.

EXAMPLE 6-12

(a) Convert the binary number 0101 to Gray code with exclusive-OR gates.

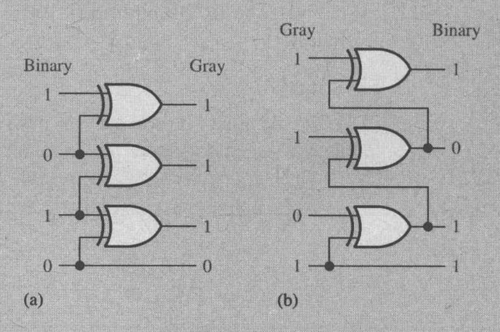
(b) Convert the Gray code 1011 to binary with exclusive-OR gates.

Solution

(a) 0101_2 is 0111 Gray. See Figure 6–40(a).

(b) 1011 Gray is 1101_2. See Figure 6–40(b).

▶ **FIGURE 6-40**



Related Problem

How many exclusive-OR gates are required to convert 8-bit binary to Gray?

SECTION 6-7 CHECKUP

1. Convert the BCD number 10000101 to binary.
2. Draw the logic diagram for converting an 8-bit binary number to Gray code.

6-8 Multiplexers (Data Selectors)

In a multiplexer, data are switched from several lines to one line.

A logic symbol for a 4-input multiplexer (MUX) is shown in Figure 6–41. Notice that there are two data-select lines because with two select bits, any one of the four data-input lines can be selected.

▶ **FIGURE 6-41**

Logic symbol for a 1-of-4 data selector/multiplexer.

MUX
Data select S_0 0, S_1 1
Data inputs D_0 0, D_1 1, D_2 2, D_3 3
Y Data output

▼ **TABLE 6-8**

Data selection for a 1-of-4-multiplexer.

Data-Select Inputs		Input Selected
S_1	S_0	
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

In Figure 6–41, a 2-bit code on the data-select (S) inputs will allow the data on the selected data input to pass through to the data output. If a binary 0 ($S_1 = 0$ and $S_0 = 0$) is applied to the data-select lines, the data on input D_0 appear on the data-output line. If a binary 1 ($S_1 = 0$ and $S_0 = 1$) is applied to the data-select lines, the data on input D_1 appear on the data output. If a binary 2 ($S_1 = 1$ and $S_0 = 0$) is applied, the data on D_2 appear on the output. If a binary 3 ($S_1 = 1$ and $S_0 = 1$) is applied, the data on D_3 are switched to the output line. A summary of this operation is given in Table 6–8.

Now let's look at the logic circuitry required to perform this multiplexing operation. The data output is equal to the state of the *selected* data input. You can therefore, derive a logic expression for the output in terms of the data input and the select inputs.

The data output is equal to D_0 only if $S_1 = 0$ and $S_0 = 0$: $Y = D_0\overline{S}_1\overline{S}_0$.

The data output is equal to D_1 only if $S_1 = 0$ and $S_0 = 1$: $Y = D_1\overline{S}_1S_0$.

The data output is equal to D_2 only if $S_1 = 1$ and $S_0 = 0$: $Y = D_2S_1\overline{S}_0$.

The data output is equal to D_3 only if $S_1 = 1$ and $S_0 = 1$: $Y = D_3S_1S_0$.

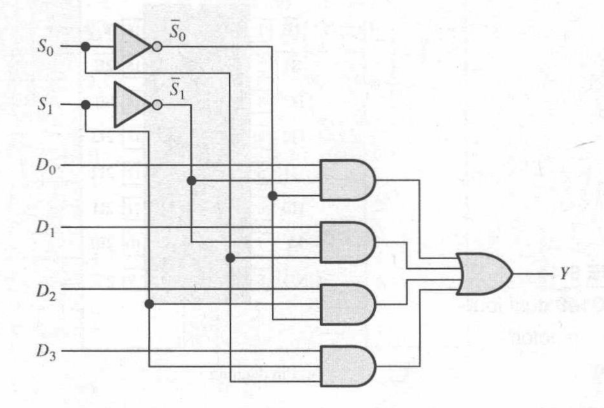
InfoNote

A *bus* is a multiple conductor pathway along which electrical signals are sent from one part of a computer to another. In computer networks, a *shared bus* is one that is connected to all the microprocessors in the system in order to exchange data. A shared bus may contain memory and input/output devices that can be accessed by all the microprocessors in the system. Access to the shared bus is controlled by a *bus arbiter* (a multiplexer of sorts) that allows only one microprocessor at a time to use the system's shared bus.

When these terms are ORed, the total expression for the data output is

$$Y = D_0\overline{S}_1\overline{S}_0 + D_1\overline{S}_1S_0 + D_2S_1\overline{S}_0 + D_3S_1S_0$$

The implementation of this equation requires four 3-input AND gates, a 4-input OR gate, and two inverters to generate the complements of S_1 and S_0, as shown in Figure 6–42. Because data can be selected from any one of the input lines, this circuit is also referred to as a **data selector**.



MultiSim

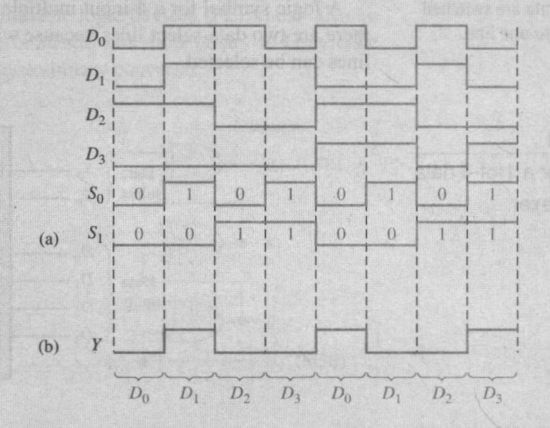
▶ **FIGURE 6-42**

Logic diagram for a 4-input multiplexer. Open file F06-44 to verify operation.

EXAMPLE 6-13

The data-input and data-select waveforms in Figure 6–43(a) are applied to the multiplexer in Figure 6–42. Determine the output waveform in relation to the inputs.

▶ **FIGURE 6-43**



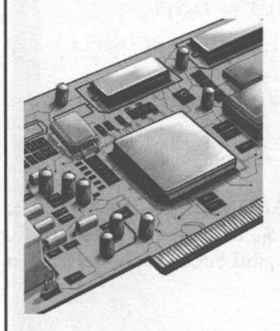
Solution

The binary state of the data-select inputs during each interval determines which data input is selected. Notice that the data-select inputs go through a repetitive binary sequence 00, 01, 10, 11, 00, 01, 10, 11, and so on. The resulting output waveform is shown in Figure 6–43(b).

Related Problem

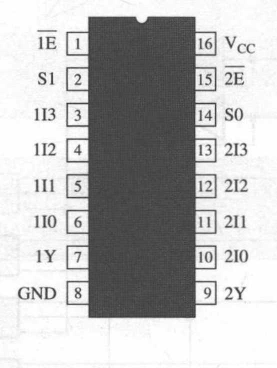
Construct a timing diagram showing all inputs and the output if the S_0 and S_1 waveforms in Figure 6–43 are interchanged.

IMPLEMENTATION: DATA SELECTOR/MULTIPLEXER

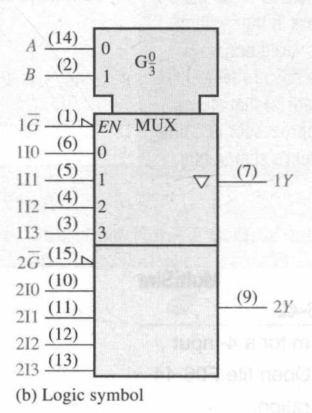


Fixed-Function Device The 74HC153 is a dual four-input data selector/multiplexer. The pin diagram is shown in Figure 6–44(a). The inputs to one of the multiplexers are 1I0 through 1I3 and the inputs to the second multiplexer are 2I0 through 2I3. The data select inputs are S0 and S1 and the active-LOW enable inputs are 1E and 2E. Each of the multiplexers has an active-LOW enable input.

The ANSI/IEEE logic symbol with dependency notation is shown in Figure 6–44(b). The two multiplexers are indicated by the partitioned outline, and the inputs common to both multiplexers are inputs to the notched block (common control block) at the top. The $G\frac{0}{3}$ dependency notation indicates an AND relationship between the two select inputs (A and B) and the inputs to each multiplexer block.



(a) Pin diagram

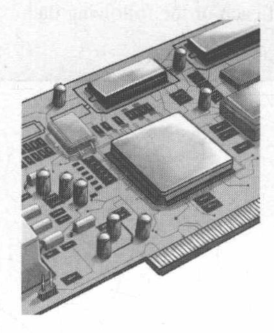


(b) Logic symbol

▶ **FIGURE 6-44**

The 74HC153 dual four-input data selector/multiplexer.

IMPLEMENTATION: EIGHT-INPUT DATA SELECTOR/MULTIPLEXER



Fixed-Function Device The 74HC151 has eight data inputs (D_0–D_7) and, therefore, three data-select or address input lines (S_0–S_2). Three bits are required to select any one of the eight data inputs ($2^3 = 8$). A LOW on the $\overline{Enable}$ input allows the selected input data to pass through to the output. Notice that the data output and its complement are both available. The pin diagram is shown in Figure 6–45(a), and the ANSI/IEEE logic symbol is shown in part (b). In this case there is no need for a common control block on the logic symbol because there is only one multiplexer to be controlled, not two as in the 74HC153. The $G\frac{0}{7}$ label within the logic symbol indicates the AND relationship between the data-select inputs and each of the data inputs 0 through 7.

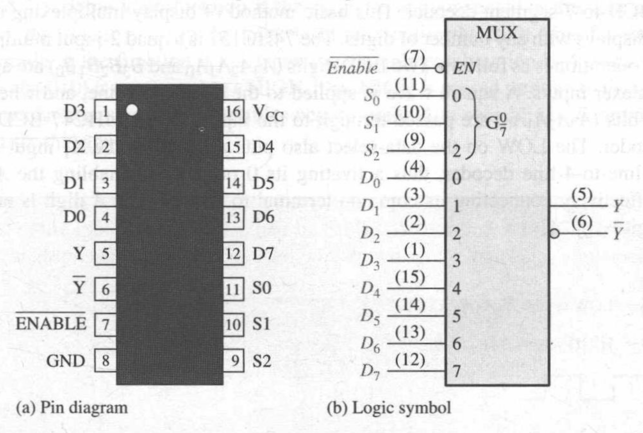


FIGURE 6-45
The 74HC151 eight-input data selector/multiplexer.

EXAMPLE 6-14

Use 74HC151s and any other logic necessary to multiplex 16 data lines onto a single data-output line.

Solution

An expansion of two 74HC151s is shown in Figure 6–46. Four bits are required to select one of 16 data inputs ($2^4 = 16$). In this application the $\overline{Enable}$ input is used as the most significant data-select bit. When the MSB in the data-select code is LOW, the left 74HC151 is enabled, and one of the data inputs (D_0 through D_7) is selected by the other three data-select bits. When the data-select MSB is HIGH, the right 74HC151 is enabled, and one of the data inputs (D_8 through D_{15}) is selected. The selected input data are then passed through to the negative-OR gate and onto the single output line.

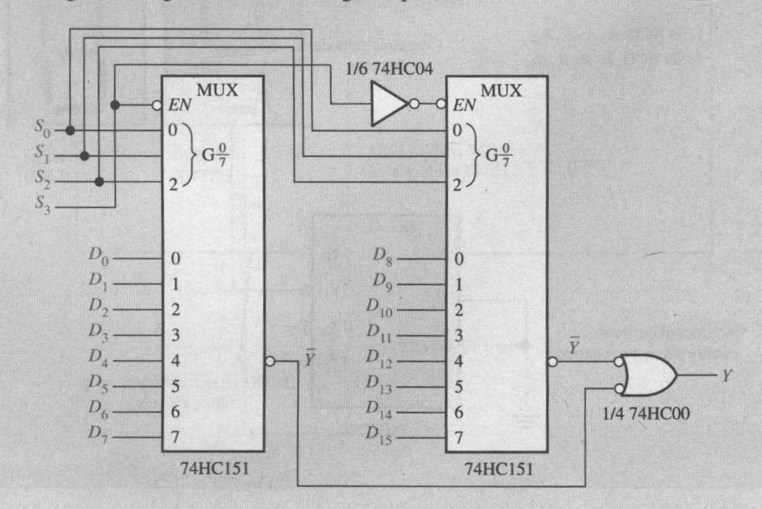


FIGURE 6-46
A 16-input multiplexer.

Related Problem

Determine the codes on the select inputs required to select each of the following data inputs: D_0, D_4, D_8, and D_{13}.

Applications

A 7-Segment Display Multiplexer

Figure 6–47 shows a simplified method of multiplexing BCD numbers to a 7-segment display. In this example, 2-digit numbers are displayed on the 7-segment readout by the use of a single BCD-to-7-segment decoder. This basic method of display multiplexing can be extended to displays with any number of digits. The 74HC157 is a quad 2-input multiplexer.

The basic operation is as follows. Two BCD digits ($A_3A_2A_1A_0$ and $B_3B_2B_1B_0$) are applied to the multiplexer inputs. A square wave is applied to the data-select line, and when it is LOW, the A bits ($A_3A_2A_1A_0$) are passed through to the inputs of the 74HC47 BCD-to-7-segment decoder. The LOW on the data-select also puts a LOW on the A_1 input of the 74HC139 2-line-to-4-line decoder, thus activating its 0 output and enabling the A-digit display by effectively connecting its common terminal to ground. The A digit is now *on* and the B digit is *off*.

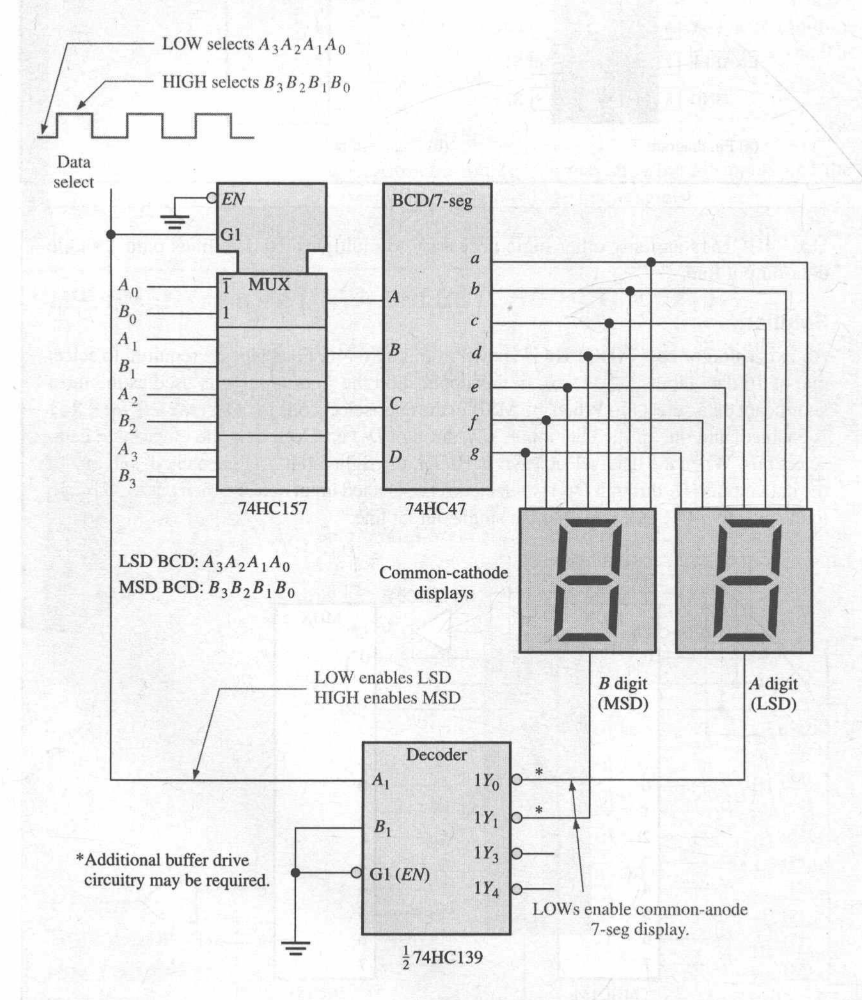


FIGURE 6-47 Simplified 7-segment display multiplexing logic.

When the data-select line goes HIGH, the B bits ($B_3B_2B_1B_0$) are passed through to the inputs of the BCD-to-7-segment decoder. Also, the 74HC139 decoder's 1 output is activated, thus enabling the B-digit display. The B digit is now *on* and the A digit is *off*. The cycle repeats at the frequency of the data-select square wave. This frequency must be high enough to prevent visual flicker as the digit displays are multiplexed.

A Logic Function Generator

A useful application of the data selector/multiplexer is in the generation of combinational logic functions in sum-of-products form. When used in this way, the device can replace discrete gates, can often greatly reduce the number of ICs, and can make design changes much easier.

To illustrate, a 74HC151 8-input data selector/multiplexer can be used to implement any specified 3-variable logic function if the variables are connected to the data-select inputs and each data input is set to the logic level required in the truth table for that function. For example, if the function is a 1 when the variable combination is $\overline{A}_2A_1\overline{A}_0$, the 2 input (selected by 010) is connected to a HIGH. This HIGH is passed through to the output when this particular combination of variables occurs on the data-select lines. Example 6–15 will help clarify this application.

EXAMPLE 6-15

Implement the logic function specified in Table 6–9 by using a 74HC151 8-input data selector/multiplexer. Compare this method with a discrete logic gate implementation.

TABLE 6-9

Inputs			Output
A_2	A_1	A_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Solution

Notice from the truth table that Y is a 1 for the following input variable combinations: 001, 011, 101, and 110. For all other combinations, Y is 0. For this function to be implemented with the data selector, the data input selected by each of the above-mentioned combinations must be connected to a HIGH (5 V). All the other data inputs must be connected to a LOW (ground), as shown in Figure 6–48.

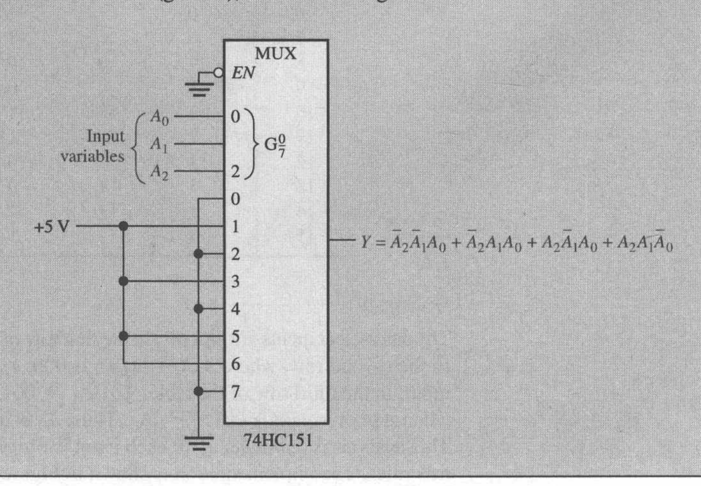


FIGURE 6-48 Data selector/multiplexer connected as a 3-variable logic function generator.

The implementation of this function with logic gates would require four 3-input AND gates, one 4-input OR gate, and three inverters unless the expression can be simplified.

Related Problem

Use the 74HC151 to implement the following expression:

$$Y = \overline{A}_2\overline{A}_1\overline{A}_0 + A_2\overline{A}_1\overline{A}_0 + \overline{A}_2A_1\overline{A}_0$$

Example 6–15 illustrated how the 8-input data selector can be used as a logic function generator for three variables. Actually, this device can be also used as a 4-variable logic function generator by the utilization of one of the bits (A_0) in conjunction with the data inputs.

A 4-variable truth table has sixteen combinations of input variables. When an 8-bit data selector is used, each input is selected twice: the first time when A_0 is 0 and the second time when A_0 is 1. With this in mind, the following rules can be applied (Y is the output, and A_0 is the least significant bit):

1. If $Y = 0$ both times a given data input is selected by a certain combination of the input variables, $A_3A_2A_1$, connect that data input to ground (0).
2. If $Y = 1$ both times a given data input is selected by a certain combination of the input variables, $A_3A_2A_1$, connect the data input to $+V$ (1).
3. If Y is different the two times a given data input is selected by a certain combination of the input variables, $A_3A_2A_1$, and if $Y = A_0$, connect that data input to A_0.
4. If Y is different the two times a given data input is selected by a certain combination of the input variables, $A_3A_2A_1$, and if $Y = \overline{A}_0$, connect that data input to $\overline{A}_0$.

EXAMPLE 6-16

Implement the logic function in Table 6–10 by using a 74HC151 8-input data selector/multiplexer. Compare this method with a discrete logic gate implementation.

▶ **TABLE 6-10**

Decimal Digit	Inputs				Output
	A_3	A_2	A_1	A_0	Y
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	1

Solution

The data-select inputs are $A_3A_2A_1$. In the first row of the table, $A_3A_2A_1 = 000$ and $Y = A_0$. In the second row, where $A_3A_2A_1$ again is 000, $Y = A_0$. Thus, A_0 is connected to the 0 input. In the third row of the table, $A_3A_2A_1 = 001$ and $Y = \overline{A}_0$. Also, in the fourth row, when $A_3A_2A_1$ again is 001, $Y = \overline{A}_0$. Thus, A_0 is inverted and connected to the 1 input. This analysis is continued until each input is properly connected according to the specified rules. The implementation is shown in Figure 6–49.

If implemented with logic gates, the function would require as many as ten 4-input AND gates, one 10-input OR gate, and four inverters, although possible simplification would reduce this requirement.

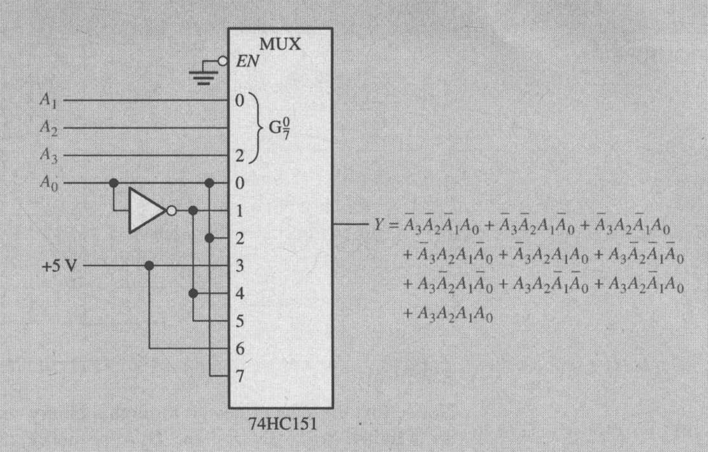


FIGURE 6-49 Data selector/multiplexer connected as a 4-variable logic function generator.

Related Problem

In Table 6–10, if $Y = 0$ when the inputs are all zeros and is alternately a 1 and a 0 for the remaining rows in the table, use a 74HC151 to implement the resulting logic function.

SECTION 6-8 CHECKUP

1. In Figure 6–42, $D_0 = 1$, $D_1 = 0$, $D_2 = 1$, $D_3 = 0$, $S_0 = 1$, and $S_1 = 0$. What is the output?
2. Identify each device.
   (a) 74HC153 (b) 74HC151
3. A 74HC151 has alternating LOW and HIGH levels on its data inputs beginning with $D_0 = 0$. The data-select lines are sequenced through a binary count (000, 001, 010, and so on) at a frequency of 1 kHz. The enable input is LOW. Describe the data output waveform.
4. Briefly describe the purpose of each of the following devices in Figure 6–47:
   (a) 74HC157 (b) 74HC47 (c) 74HC139

6-9 Demultiplexers

In a demultiplexer, data are switched from one line to several lines.

Figure 6–50 shows a 1-line-to-4-line demultiplexer (DEMUX) circuit. The data-input line goes to all of the AND gates. The two data-select lines enable only one gate at a time, and the data appearing on the data-input line will pass through the selected gate to the associated data-output line.

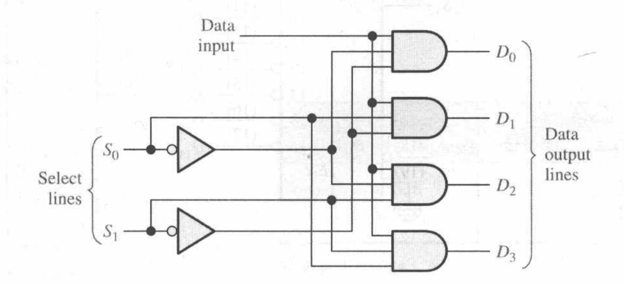
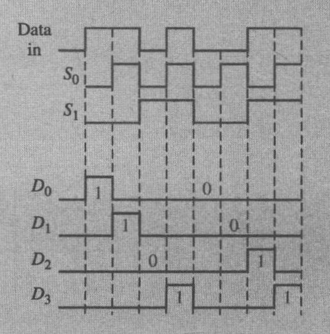


FIGURE 6-50 A 1-line-to-4-line demultiplexer.

EXAMPLE 6-17

The serial data-input waveform (Data in) and data-select inputs (S_0 and S_1) are shown in Figure 6–51. Determine the data-output waveforms on D_0 through D_3 for the demultiplexer in Figure 6–50.



▶ **FIGURE 6-51**

Solution

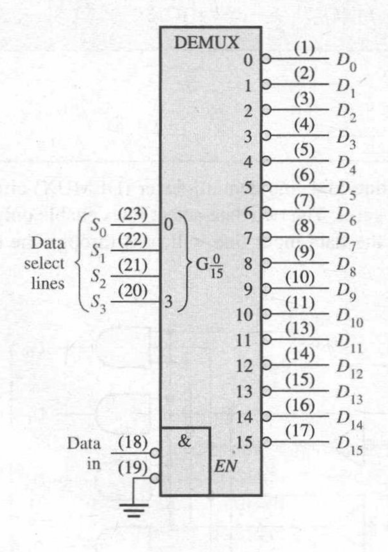
Notice that the select lines go through a binary sequence so that each successive input bit is routed to D_0, D_1, D_2, and D_3 in sequence, as shown by the output waveforms in Figure 6–51.

Related Problem

Develop the timing diagram for the demultiplexer if the S_0 and S_1 waveforms are both inverted.

4-Line-to-16-Line Decoder as a Demultiplexer

We have already discussed a 4-line-to-16-line decoder (Section 6–5). This device and other decoders can also be used in demultiplexing applications. The logic symbol for this device when used as a demultiplexer is shown in Figure 6–52. In demultiplexer applications, the input lines are used as the data-select lines. One of the chip select inputs is used as the data-input line, with the other chip select input held LOW to enable the internal negative-AND gate at the bottom of the diagram.



◀ **FIGURE 6-52**
The decoder used as a demultiplexer.

SECTION 6-9 CHECKUP

1. Generally, how can a decoder be used as a demultiplexer?
2. The demultiplexer in Figure 6–52 has a binary code of 1010 on the data-select lines, and the data-input line is LOW. What are the states of the output lines?

6-10 Parity Generators/Checkers

The parity method of error detection in which a **parity bit** is attached to a group of information bits in order to make the total number of 1s either even or odd (depending on the system) was covered in Chapter 2. In addition to parity bits, several specific codes also provide inherent error detection.

Basic Parity Logic

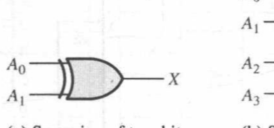
A parity bit indicates if the number of 1s in a code is even or odd for the purpose of error detection.

In order to check for or to generate the proper parity in a given code, a basic principle can be used:

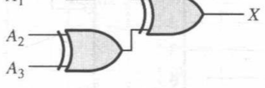
The sum (disregarding carries) of an even number of 1s is always 0, and the sum of an odd number of 1s is always 1.

Therefore, to determine if a given code has **even parity** or **odd parity,** all the bits in that code are summed. As you know, the modulo-2 sum of two bits can be generated by an exclusive-OR gate, as shown in Figure 6–53(a); the modulo-2 sum of four bits can be formed by three exclusive-OR gates connected as shown in Figure 6–53(b); and so on. When the number of 1s on the inputs is even, the output X is 0 (LOW). When the number of 1s is odd, the output X is 1 (HIGH).

FIGURE 6-53

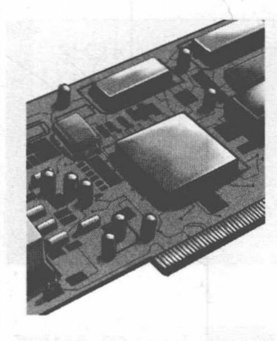


(a) Summing of two bits

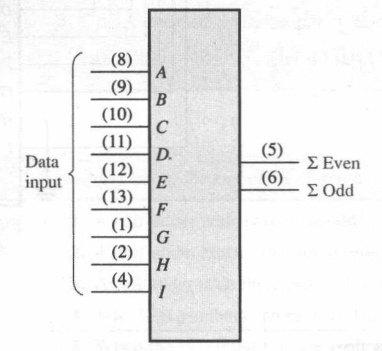


(b) Summing of four bits

IMPLEMENTATION: 9-BIT PARITY GENERATOR/CHECKER



Fixed-Function Device The logic symbol and function table for a 74HC280 are shown in Figure 6–54. This particular device can be used to check for odd or even parity on a 9-bit code (eight data bits and one parity bit), or it can be used to generate a parity bit for a binary code with up to nine bits. The inputs are A through I; when there is an even number of 1s on the inputs, the Σ Even output is HIGH and the Σ Odd output is LOW.



(a) Traditional logic symbol

Number of Inputs A–I that Are High	Outputs	
	Σ Even	Σ Odd
0, 2, 4, 6, 8	H	L
1, 3, 5, 7, 9	L	H

(b) Function table

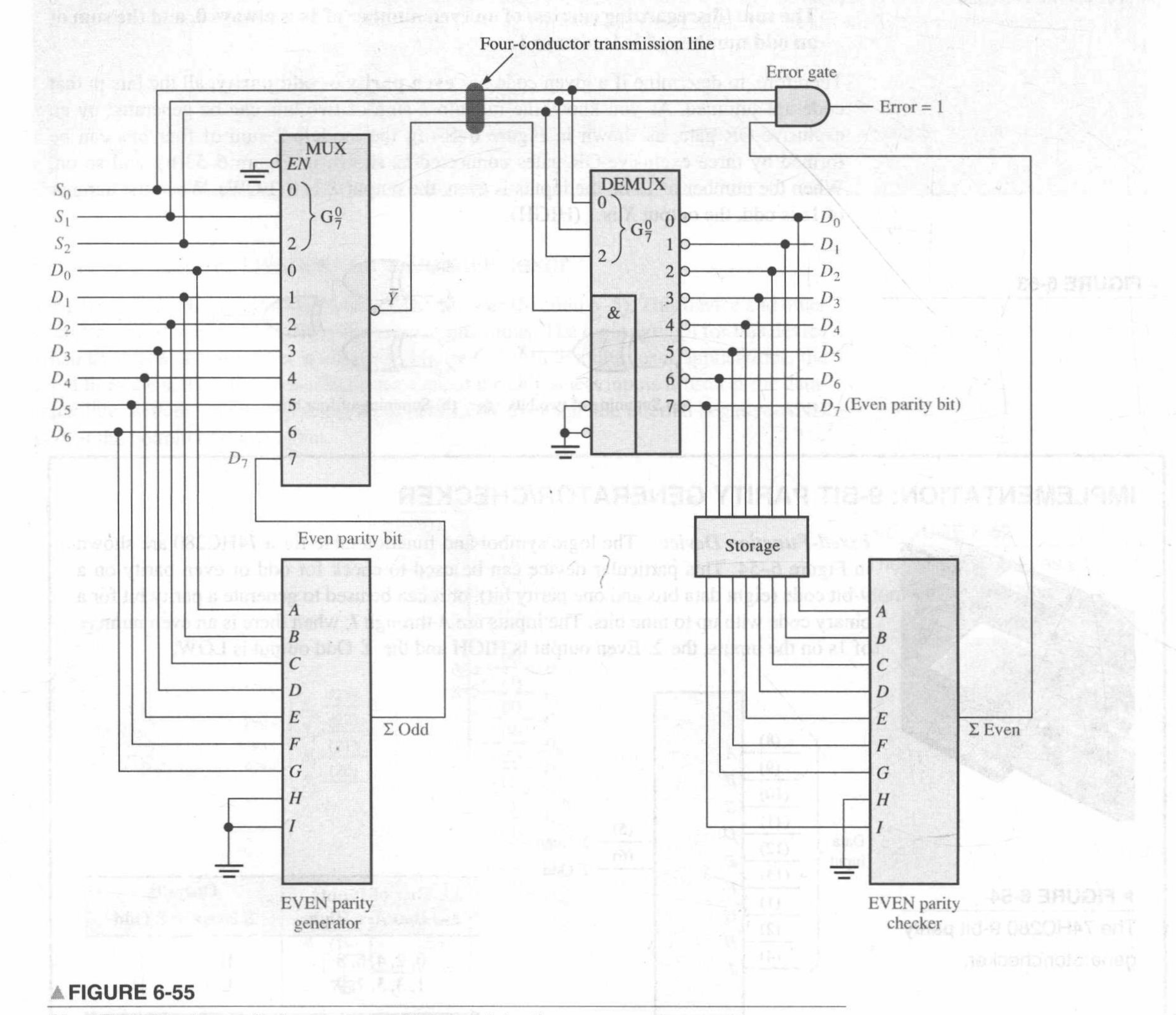
FIGURE 6-54 The 74HC280 9-bit parity generator/checker.

Parity Checker When this device is used as an even parity checker, the number of input bits should always be even; and when a parity error occurs, the Σ Even output goes LOW and the Σ Odd output goes HIGH. When it is used as an odd parity checker, the number of input bits should always be odd; and when a parity error occurs, the Σ Odd output goes LOW and the Σ Even output goes HIGH.

Parity Generator If this device is used as an even parity generator, the parity bit is taken at the Σ Odd output because this output is a 0 if there is an even number of input bits and it is a 1 if there is an odd number. When used as an odd parity generator, the parity bit is taken at the Σ Even output because it is a 0 when the number of inputs bits is odd.

A Data Transmission System with Error Detection

A simplified data transmission system is shown in Figure 6–55 to illustrate an application of parity generators/checkers, as well as multiplexers and demultiplexers, and to illustrate the need for data storage in some applications.



▲ FIGURE 6-55

Simplified data transmission system with error detection.

In this application, digital data from seven sources are multiplexed onto a single line for transmission to a distant point. The seven data bits (D_0 through D_6) are applied to the multiplexer data inputs and, at the same time, to the even parity generator inputs. The Σ Odd output of the parity generator is used as the even parity bit. This bit is 0 if the number of 1s on the inputs A through I is even and is a 1 if the number of 1s on A through I is odd. This bit is D_7 of the transmitted code.

The data-select inputs are repeatedly cycled through a binary sequence, and each data bit, beginning with D_0, is serially passed through and onto the transmission line ($\overline{Y}$). In this example, the transmission line consists of four conductors: one carries the serial data and three carry the timing signals (data selects). There are more sophisticated ways of sending the timing information, but we are using this direct method to illustrate a basic principle.

InfoNote

Microprocessors perform internal parity checks as well as parity checks of the external data and address buses. In a read operation, the external system can transfer the parity information together with the data bytes. The microprocessor checks whether the resulting parity is even and sends out the corresponding signal. When it sends out an address code, the microprocessor does not perform an address parity check, but it does generate an even parity bit for the address.

At the demultiplexer end of the system, the data-select signals and the serial data stream are applied to the demultiplexer. The data bits are distributed by the demultiplexer onto the output lines in the order in which they occurred on the multiplexer inputs. That is, D_0 comes out on the D_0 output, D_1 comes out on the D_1 output, and so on. The parity bit comes out on the D_7 output. These eight bits are temporarily stored and applied to the even parity checker. Not all of the bits are present on the parity checker inputs until the parity bit D_7 comes out and is stored. At this time, the error gate is enabled by the data-select code 111. If the parity is correct, a 0 appears on the Σ Even output, keeping the Error output at 0. If the parity is incorrect, all 1s appear on the error gate inputs, and a 1 on the Error output results.

This particular application has demonstrated the need for data storage. Storage devices will be introduced in Chapter 7 and covered in Chapter 10.

The timing diagram in Figure 6–56 illustrates a specific case in which two 8-bit words are transmitted, one with correct parity and one with an error.

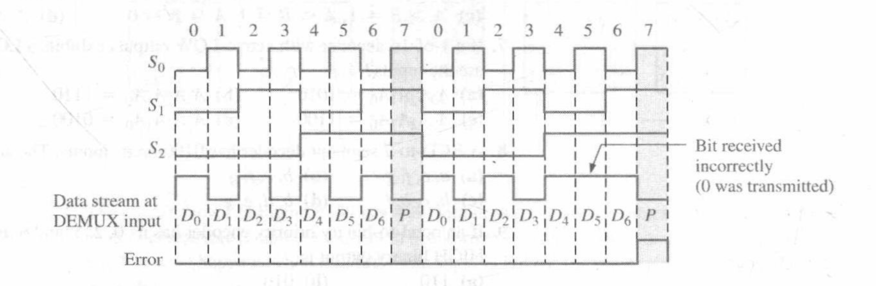


FIGURE 6-56 Example of data transmission with and without error for the system in Figure 6–55.

SECTION 6-10 CHECKUP

1. Add an even parity bit to each of the following codes:
   (a) 110100 (b) 01100011
2. Add an odd parity bit to each of the following codes:
   (a) 1010101 (b) 1000001
3. Check each of the even parity codes for an error.
   (a) 100010101 (b) 1110111001

TRUE/FALSE QUIZ

Answers are at the end of the chapter.

1. A half-adder adds two binary bits.
2. A half-adder has a sum output only.
3. A full-adder adds three bits and produces two outputs.
4. Two 4-bit numbers can be added using two full-adders.
5. When the two input bits are both 1 and the carry input bit is a 1, the sum output of a full adder is 0.
6. A comparator determines when two binary numbers are equal.

7. A decoder detects the presence of a specified combination of input bits.

8. The 4-line-to-10-line decoder and the 1-of-10 decoder are two different types.

9. An encoder essentially performs a reverse decoder function.

10. A multiplexer is a logic circuit that allows digital information from a single source to be routed onto several lines.

SELF-TEST

Answers are at the end of the chapter.

1. A half-adder is characterized by
(a) two inputs and two outputs (b) three inputs and two outputs
(c) two inputs and three outputs (d) two inputs and one output

2. A full-adder is characterized by
(a) two inputs and two outputs (b) three inputs and two outputs
(c) two inputs and three outputs (d) two inputs and one output

3. The inputs to a full-adder are $A = 1, B = 1, C_{in} = 0$. The outputs are
(a) $\Sigma = 1, C_{out} = 1$ (b) $\Sigma = 1, C_{out} = 0$
(c) $\Sigma = 0, C_{out} = 1$ (d) $\Sigma = 0, C_{out} = 0$

4. A 4-bit parallel adder can add
(a) two 4-bit binary numbers (b) two 2-bit binary numbers
(c) four bits at a time (d) four bits in sequence

5. To expand a 4-bit parallel adder to an 8-bit parallel adder, you must
(a) use four 4-bit adders with no interconnections
(b) use two 4-bit adders and connect the sum outputs of one to the bit inputs of the other
(c) use eight 4-bit adders with no interconnections
(d) use two 4-bit adders with the carry output of one connected to the carry input of the other

6. If a 74HC85 magnitude comparator has $A = 1011$ and $B = 1001$ on its inputs, the outputs are
(a) $A > B = 0, A < B = 1, A = B = 0$ (b) $A > B = 1, A < B = 0, A = B = 0$
(c) $A > B = 1, A < B = 1, A = B = 0$ (d) $A > B = 0, A < B = 0, A = B = 1$

7. If a 1-of-16 decoder with active-LOW outputs exhibits a LOW on the decimal 12 output, what are the inputs?
(a) $A_3A_2A_1A_0 = 1010$ (b) $A_3A_2A_1A_0 = 1110$
(c) $A_3A_2A_1A_0 = 1100$ (d) $A_3A_2A_1A_0 = 0100$

8. A BCD-to-7 segment decoder has 0100 on its inputs. The active outputs are
(a) *a, c, f, g* (b) *b, c, f, g*
(c) *b, c, e, f* (d) *b, d, e, g*

9. If an octal-to-binary priority encoder has its 0, 2, 5, and 6 inputs at the active level, the active-HIGH binary output is
(a) 110 (b) 010
(c) 101 (d) 000

10. In general, a multiplexer has
(a) one data input, several data outputs, and selection inputs
(b) one data input, one data output, and one selection input
(c) several data inputs, several data outputs, and selection inputs
(d) several data inputs, one data output, and selection inputs

11. Data selectors are basically the same as
(a) decoders (b) demultiplexers
(c) multiplexers (d) encoders

12. Which of the following codes exhibit even parity?
(a) 10011000 (b) 01111000
(c) 11111111 (d) 11010101
(e) all (f) both answers (b) and (c)

PROBLEMS

Answers to odd-numbered problems are at the end of the book.

Section 6–1 Half and Full Adders

1. For the full-adder of Figure 6–4, determine the logic state (1 or 0) at each gate output for the following inputs:
(a) $A = 1, B = 1, C_{in} = 1$ (b) $A = 0, B = 1, C_{in} = 1$
(c) $A = 0, B = 1, C_{in} = 0$

2. What are the full-adder inputs that will produce each of the following outputs:

   (a) $\Sigma = 0, C_{out} = 0$ (b) $\Sigma = 1, C_{out} = 0$
   (c) $\Sigma = 1, C_{out} = 1$ (d) $\Sigma = 0, C_{out} = 1$

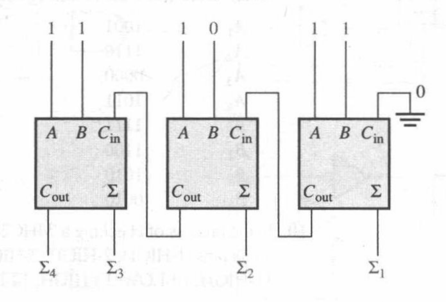
3. Determine the outputs of a full-adder for each of the following inputs:

   (a) $A = 1, B = 0, C_{in} = 0$ (b) $A = 0, B = 0, C_{in} = 1$
   (c) $A = 0, B = 1, C_{in} = 1$ (d) $A = 1, B = 1, C_{in} = 1$

Section 6–2 Parallel Binary Adders

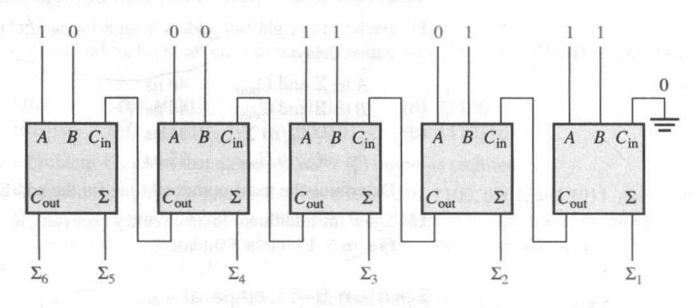
4. For the parallel adder in Figure 6–57, determine the complete sum by analysis of the logical operation of the circuit. Verify your result by longhand addition of the two input numbers.

FIGURE 6-57



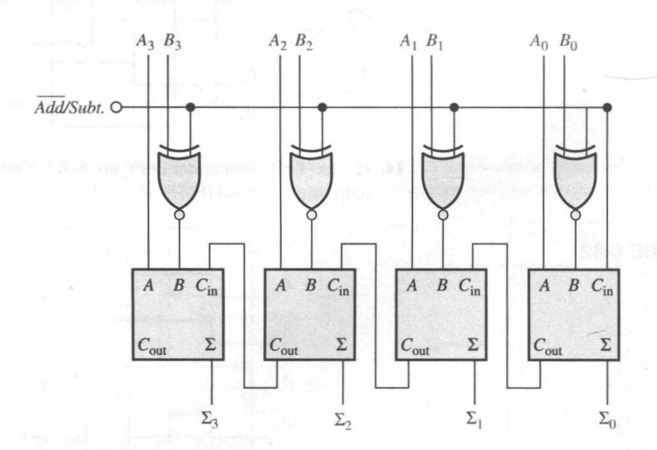
5. Repeat Problem 4 for the circuit and input conditions in Figure 6–58.

FIGURE 6-58



6. The circuit shown in Figure 6–59 is a 4-bit circuit that can add or subtract numbers in a form used in computers (positive numbers in true form; negative numbers in complement form). (a) Explain what happens when the $\overline{Add}/Subt.$ input is HIGH. (b) What happens when $\overline{Add}/Subt.$ is LOW?

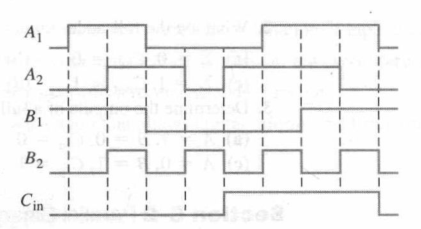
FIGURE 6-59



7. For the circuit in Figure 6–59, assume the inputs are $\overline{Add}/Subt. = 1$, $A = 1001$ and $B = 1100$. What is the output?

8. The input waveforms in Figure 6–60 are applied to a 2-bit adder. Determine the waveforms for the sum and the output carry in relation to the inputs by constructing a timing diagram.

▶ FIGURE 6-60



9. The following sequences of bits (right-most bit first) appear on the inputs to a 4-bit parallel adder. Determine the resulting sequence of bits on each sum output.

A_1	1001
A_2	1110
A_3	0000
A_4	1011
B_1	1111
B_2	1100
B_3	1010
B_4	0010

10. In the process of checking a 74HC283 4-bit parallel adder, the following logic levels are observed on its pins: 1-HIGH, 2-HIGH, 3-HIGH, 4-HIGH, 5-LOW, 6-LOW, 7-LOW, 9-HIGH, 10-LOW, 11-HIGH, 12-LOW, 13-HIGH, 14-HIGH, and 15-HIGH. Determine if the IC is functioning properly.

Section 6–3 Ripple Carry and Look-Ahead Carry Adders

11. Each of the eight full-adders in an 8-bit parallel ripple carry adder exhibits the following propagation delays:

A to Σ and C_{out}:	40 ns
B to Σ and C_{out}:	40 ns
C_{in} to Σ:	35 ns
C_{in} to C_{out}:	25 ns

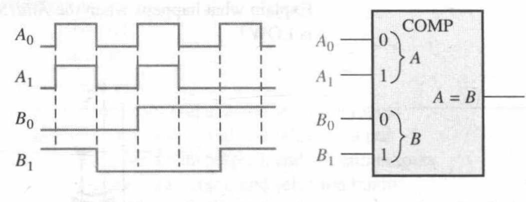
Determine the maximum total time for the addition of two 8-bit numbers.

12. Show the additional logic circuitry necessary to make the 4-bit look-ahead carry adder in Figure 6–17 into a 5-bit adder.

Section 6–4 Comparators

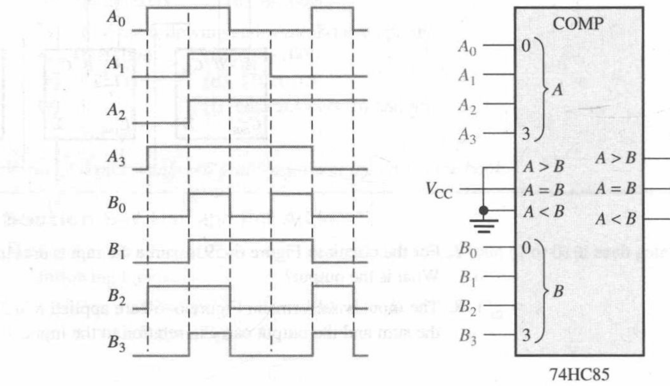
13. The waveforms in Figure 6–61 are applied to the comparator as shown. Determine the output ($A = B$) waveform.

▶ FIGURE 6-61



14. For the 4-bit comparator in Figure 6–62, plot each output waveform for the inputs shown. The outputs are active-HIGH.

▶ FIGURE 6-62



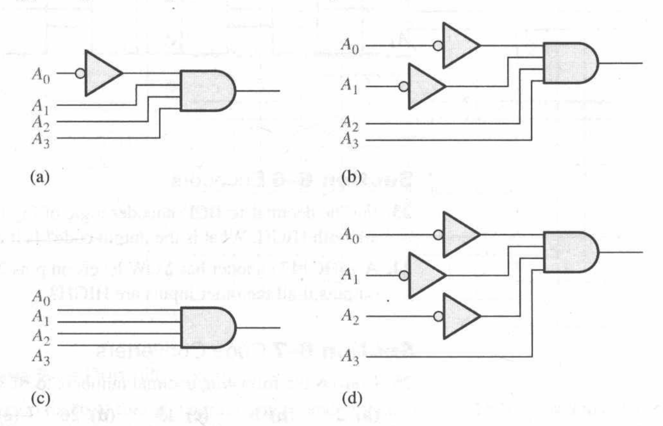
15. For each set of binary numbers, determine the output states for the comparator of Figure 6–21.

(a) $A_3A_2A_1A_0 = 1100$, $B_3B_2B_1B_0 = 1001$ **(b)** $A_3A_2A_1A_0 = 1000$, $B_3B_2B_1B_0 = 1011$ **(c)** $A_3A_2A_1A_0 = 0100$, $B_3B_2B_1B_0 = 0100$

Section 6–5 Decoders

16. When a HIGH is on the output of each of the decoding gates in Figure 6–63, what is the binary code appearing on the inputs? The MSB is A_3.

FIGURE 6-63



17. Show the decoding logic for each of the following codes if an active-HIGH (1) output is required:

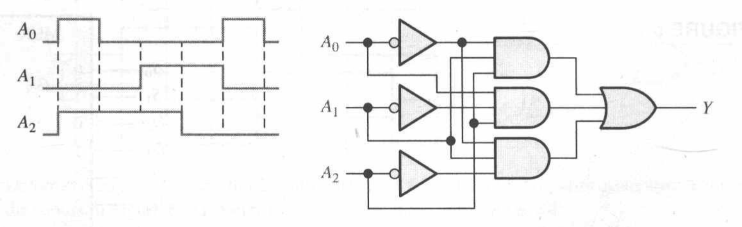
(a) 1101 **(b)** 1000 **(c)** 11011 **(d)** 11100
(e) 101010 **(f)** 111110 **(g)** 000101 **(h)** 1110110

18. Solve Problem 17, given that an active-LOW (0) output is required.

19. You wish to detect only the presence of the codes 1010, 1100, 0001, and 1011. An active-HIGH output is required to indicate their presence. Develop the minimum decoding logic with a single output that will indicate when any one of these codes is on the inputs. For any other code, the output must be LOW.

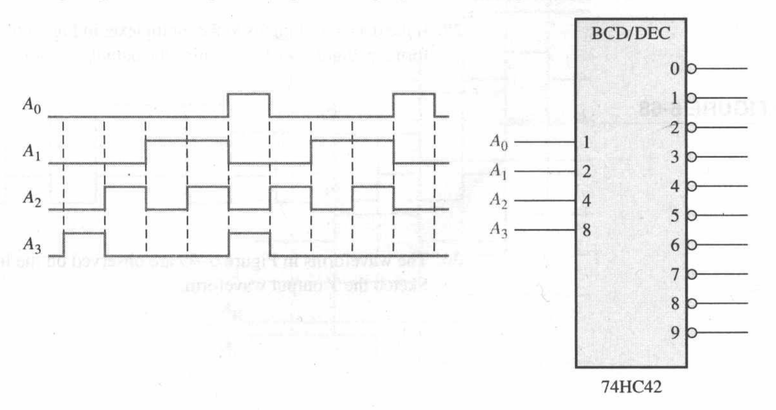
20. If the input waveforms are applied to the decoding logic as indicated in Figure 6–64, sketch the output waveform in proper relation to the inputs.

FIGURE 6-64



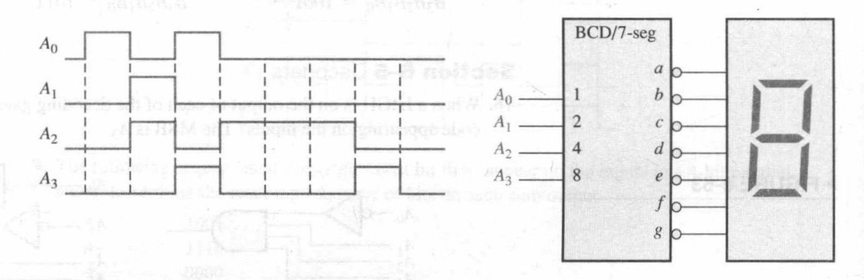
21. BCD numbers are applied sequentially to the BCD-to-decimal decoder in Figure 6–65. Draw a timing diagram, showing each output in the proper relationship with the others and with the inputs.

FIGURE 6-65



22. A 7-segment decoder/driver drives the display in Figure 6–66. If the waveforms are applied as indicated, determine the sequence of digits that appears on the display.

▶ FIGURE 6-66



Section 6–6 Encoders

23. For the decimal-to-BCD encoder logic of Figure 6–35, assume that the 9 input and the 3 input are both HIGH. What is the output code? Is it a valid BCD (8421) code?

24. A 74HC147 encoder has LOW levels on pins 2, 5, and 12. What BCD code appears on the outputs if all the other inputs are HIGH?

Section 6–7 Code Converters

25. Convert the following decimal numbers to BCD and then to binary.

(a) 2 **(b)** 8 **(c)** 13 **(d)** 26 **(e)** 33

26. Show the logic required to convert a 10-bit binary number to Gray code, and use that logic to convert the following binary numbers to Gray code:

(a) 1010101010 **(b)** 1111100000 **(c)** 0000001110 **(d)** 1111111111

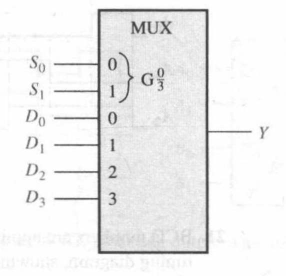
27. Show the logic required to convert a 10-bit Gray code to binary, and use that logic to convert the following Gray code words to binary:

(a) 1010000000 **(b)** 0011001100 **(c)** 1111000111 **(d)** 0000000001

Section 6–8 Multiplexers (Data Selectors)

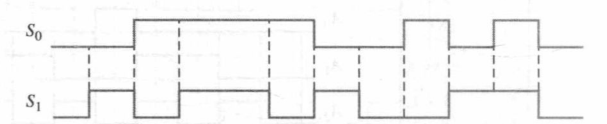
28. For the multiplexer in Figure 6–67, determine the output for the following input states: $D_0 = 0$, $D_1 = 1, D_2 = 1, D_3 = 0, S_0 = 1, S_1 = 0$.

▶ FIGURE 6-67



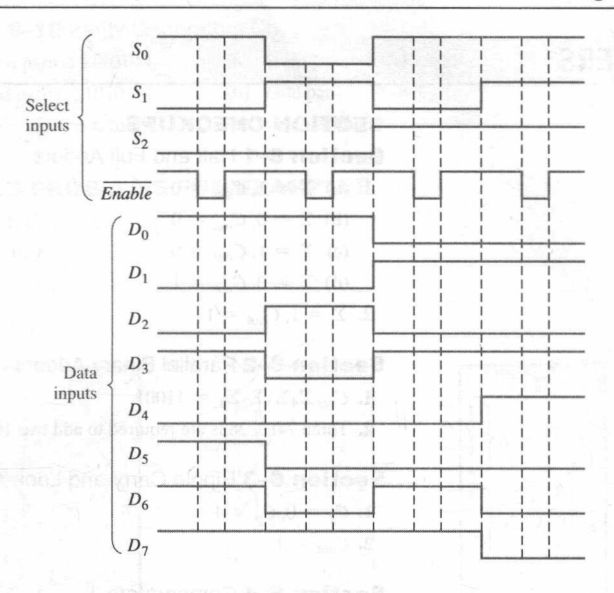
29. If the data-select inputs to the multiplexer in Figure 6–67 are sequenced as shown by the waveforms in Figure 6–68, determine the output waveform with the data inputs specified in Problem 28.

▶ FIGURE 6-68



30. The waveforms in Figure 6–69 are observed on the inputs of a 74HC151 8-input multiplexer. Sketch the Y output waveform.

▶ FIGURE 6-69



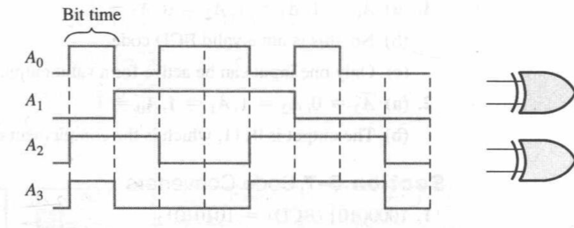
Section 6–9 Demultiplexers

31. Develop the total timing diagram (inputs and outputs) for a 74HC154 used in a demultiplexing application in which the inputs are as follows: The data-select inputs are repetitively sequenced through a straight binary count beginning with 0000, and the data input is a serial data stream carrying BCD data representing the decimal number 2468. The least significant digit (8) is first in the sequence, with its LSB first, and it should appear in the first 4-bit positions of the output.

Section 6–10 Parity Generators/Checkers

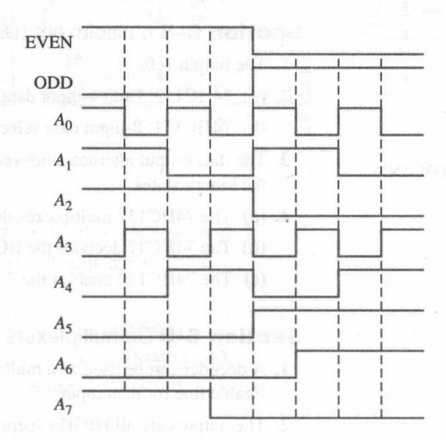
32. The waveforms in Figure 6–70 are applied to the 4-bit parity logic. Determine the output waveform in proper relation to the inputs. For how many bit times does even parity occur, and how is it indicated? The timing diagram includes eight bit times.

▶ FIGURE 6-70



33. Determine the Σ Even and the Σ Odd outputs of a 74HC280 9-bit parity generator/checker for the inputs in Figure 6–71. Refer to the function table in Figure 6–54.

▶ FIGURE 6-71



ANSWERS

SECTION CHECKUPS

Section 6–1 Half and Full Adders

1. (a) $\Sigma = 1, C_{out} = 0$
   (b) $\Sigma = 0, C_{out} = 0$
   (c) $\Sigma = 1, C_{out} = 0$
   (d) $\Sigma = 0, C_{out} = 1$
2. $\Sigma = 1, C_{out} = 1$

Section 6–2 Parallel Binary Adders

1. $C_{out}\Sigma_4\Sigma_3\Sigma_2\Sigma_1 = 11001$
2. Three 74HC283s are required to add two 10-bit numbers.

Section 6–3 Ripple Carry and Look-Ahead Carry Adders

1. $C_g = 0, C_p = 1$
2. $C_{out} = 1$

Section 6–4 Comparators

1. $A > B = 1, A < B = 0, A = B = 0$ when $A = 1011$ and $B = 1010$
2. Right comparator: $A < B = 1; A = B = 0; A > B = 0$
   Left comparator: $A < B = 0; A = B = 0; A > B = 1$

Section 6–5 Decoders

1. Output 5 is active when 101 is on the inputs.
2. Four 74HC154s are used to decode a 6-bit binary number.
3. Active-HIGH output drives a common-cathode LED display.

Section 6–6 Encoders

1. (a) $A_0 = 1, A_1 = 1, A_2 = 0, A_3 = 1$
   (b) No, this is not a valid BCD code.
   (c) Only one input can be active for a valid output.
2. (a) $\overline{A}_3 = 0, \overline{A}_2 = 1, \overline{A}_1 = 1, \overline{A}_0 = 1$
   (b) The output is 0111, which is the complement of 1000 (8).

Section 6–7 Code Converters

1. 10000101 (BCD) $= 1010101_2$
2. An 8-bit binary-to-Gray converter consists of seven exclusive-OR gates in an arrangement like that in Figure 6–40 but with inputs B_0–B_7.

Section 6–8 Multiplexers (Data Selectors)

1. The output is 0.
2. (a) 74HC153: Dual 4-input data selector/multiplexer
   (b) 74HC151: 8-input data selector/multiplexer
3. The data output alternates between LOW and HIGH as the data-select inputs sequence through the binary states.
4. (a) The 74HC157 multiplexes the two BCD codes to the 7-segment decoder.
   (b) The 74HC47 decodes the BCD to energize the display.
   (c) The 74HC139 enables the 7-segment displays alternately.

Section 6–9 Demultiplexers

1. A decoder can be used as a multiplexer by using the input lines for data selection and an Enable line for data input.
2. The outputs are all HIGH except D_{10}, which is LOW.

Section 6–10 Parity Generators/Checkers

1. (a) Even parity: <u>1</u>110100 (b) Even parity: <u>0</u>01100011
2. (a) Odd parity: <u>1</u>1010101 (b) Odd parity: <u>1</u>1000001
3. (a) Code is correct, four 1s. (b) Code is in error, seven 1s

RELATED PROBLEMS FOR EXAMPLES

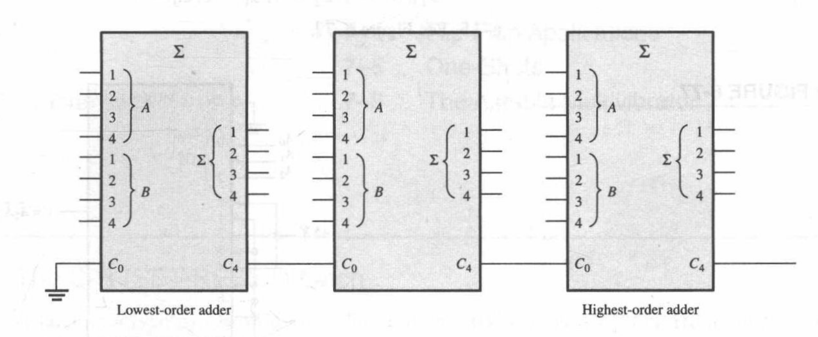
6–1 $\Sigma = 1, C_{out} = 1$

6–2 $\Sigma_1 = 0, \Sigma_2 = 0, \Sigma_3 = 1, \Sigma_4 = 1$

6–3 1011 + 1010 = 10101

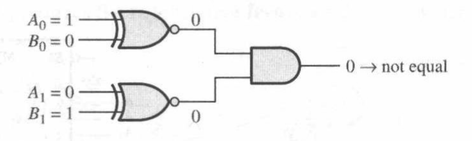
6–4 See Figure 6–72.

FIGURE 6-72



6–5 See Figure 6–73.

FIGURE 6-73

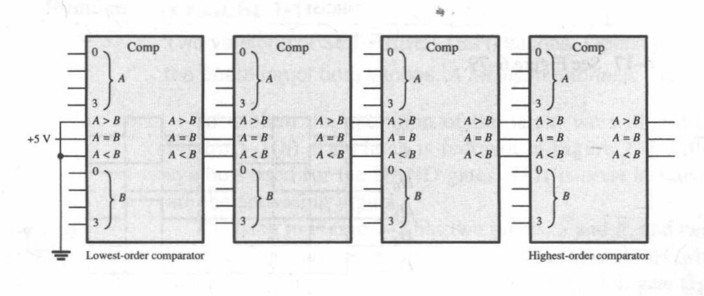


6–6 $A > B = 0, A = B = 0, A < B = 1$

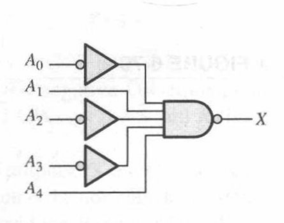
6–7 See Figure 6–74.

6–8 See Figure 6–75.

6–9 Output 22



▲FIGURE 6-74

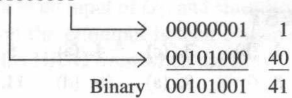


▲FIGURE 6-75

6–10 All inputs LOW: $\overline{A}_0 = 0, \overline{A}_1 = 1, \overline{A}_2 = 1, \overline{A}_3 = 0$

All inputs HIGH: All outputs HIGH.

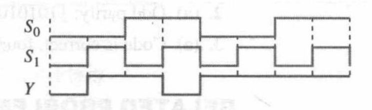
6–11 BCD 01000001



6–12 Seven exclusive-OR gates

6–13 See Figure 6–76.

► **FIGURE 6-76**



6–14 D_0: $S_3 = 0, S_2 = 0, S_1 = 0, S_0 = 0$

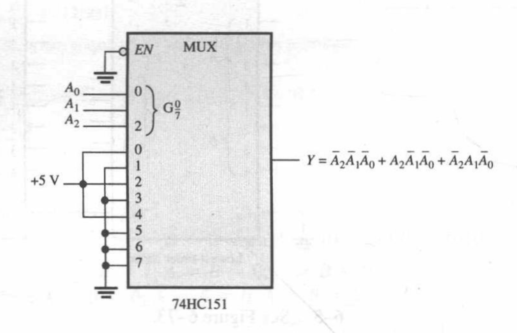
D_4: $S_3 = 0, S_2 = 1, S_1 = 0, S_0 = 0$

D_8: $S_3 = 1, S_2 = 0, S_1 = 0, S_0 = 0$

D_{13}: $S_3 = 1, S_2 = 1, S_1 = 0, S_0 = 1$

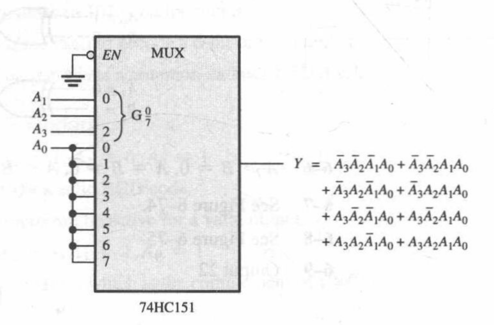
6–15 See Figure 6–77.

► **FIGURE 6-77**



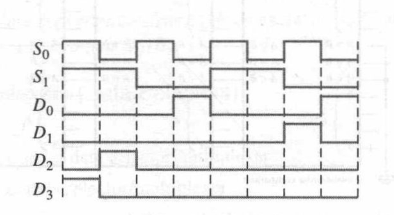
6–16 See Figure 6–78.

► **FIGURE 6-78**



6–17 See Figure 6–79.

► **FIGURE 6-79**



TRUE/FALSE QUIZ

1. T **2.** F **3.** T **4.** F **5.** F
6. T **7.** T **8.** F **9.** T **10.** F

SELF-TEST

1. (a) **2.** (b) **3.** (c) **4.** (a) **5.** (d) **6.** (b)
7. (c) **8.** (b) **9.** (a) **10.** (d) **11.** (c) **12.** (f)

Chapter 7 Latches, Flip–Flops, and Timers

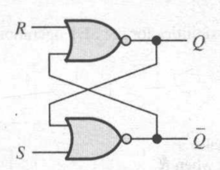
CHAPTER OUTLINE

7-1 Latches

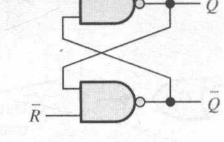
> **InfoNote**
>
> Latches are sometimes used for multiplexing data onto a bus. For example, data being input to a computer from an external source have to share the data bus with data from other sources. When the data bus becomes unavailable to the external source, the existing data must be temporarily stored, and latches placed between the external source and the data bus may be used to do this.

The S-R (SET-RESET) Latch

A latch is a type of **bistable** logic device or **multivibrator.** An active-HIGH input S-R (SET-RESET) latch is formed with two cross-coupled NOR gates, as shown in Figure 7–1(a); an active-LOW input $\overline{S}$-$\overline{R}$ latch is formed with two cross-coupled NAND gates, as shown in Figure 7–1(b). Notice that the output of each gate is connected to an input of the opposite gate. This produces the regenerative **feedback** that is characteristic of all latches and flip-flops.



(a) Active-HIGH input S-R latch



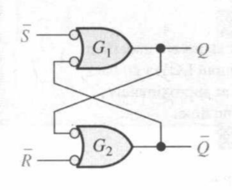
(b) Active-LOW input $\overline{S}$-$\overline{R}$ latch

MultiSim

▲ **FIGURE 7-1**

Two versions of SET-RESET (S-R) latches. Open files F07-01(a) and (b) and verify the operation of both latches. *A Multisim tutorial is available on the website.*

To explain the operation of the latch, we will use the NAND gate $\overline{S}$-$\overline{R}$ latch in Figure 7–1(b). This latch is redrawn in Figure 7–2 with the negative-OR equivalent symbols used for the NAND gates. This is done because LOWs on the $\overline{S}$ and $\overline{R}$ lines are the activating inputs.



▲ **FIGURE 7-2**

Negative-OR equivalent of the NAND gate $\overline{S}$-$\overline{R}$ latch in Figure 7–1(b).

The latch in Figure 7–2 has two inputs, $\overline{S}$ and $\overline{R}$, and two outputs, Q and $\overline{Q}$. Let's start by assuming that both inputs and the Q output are HIGH, which is the normal latched state. Since the Q output is connected back to an input of gate G_2, and the $\overline{R}$ input is HIGH, the output of G_2 must be LOW. This LOW output is coupled back to an input of gate G_1, ensuring that its output is HIGH.

When the Q output is HIGH, the latch is in the **SET** state. It will remain in this state indefinitely until a LOW is temporarily applied to the $\overline{R}$ input. With a LOW on the $\overline{R}$ input and a HIGH on $\overline{S}$, the output of gate G_2 is forced HIGH. This HIGH on the $\overline{Q}$ output is coupled back to an input of G_1, and since the $\overline{S}$ input is HIGH, the output of G_1 goes LOW. This LOW on the Q output is then coupled back to an input of G_2, ensuring that the $\overline{Q}$ output remains HIGH even when the LOW on the $\overline{R}$ input is removed. When the Q output

is LOW, the latch is in the **RESET** state. Now the latch remains indefinitely in the RESET state until a momentary LOW is applied to the $\overline{S}$ input.

A latch can reside in either of its two

In normal operation, the outputs of a latch are always complements of each other.

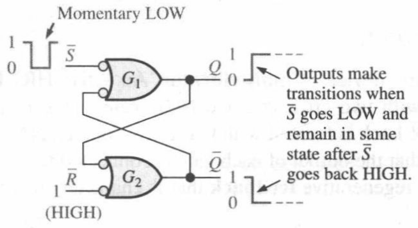
When Q is HIGH, $\overline{Q}$ is LOW, and when Q is LOW, $\overline{Q}$ is HIGH.

An invalid condition in the operation of an active-LOW input $\overline{S}$-$\overline{R}$ latch occurs when LOWs are applied to both $\overline{S}$ and $\overline{R}$ at the same time. As long as the LOW levels are simultaneously held on the inputs, both the Q and $\overline{Q}$ outputs are forced HIGH, thus violating the basic complementary operation of the outputs. Also, if the LOWs are released simultaneously, both outputs will attempt to go LOW. Since there is always some small difference in the propagation delay time of the gates, one of the gates will dominate in its transition to the LOW output state. This, in turn, forces the output of the slower gate to remain HIGH. In this situation, you cannot reliably predict the next state of the latch.

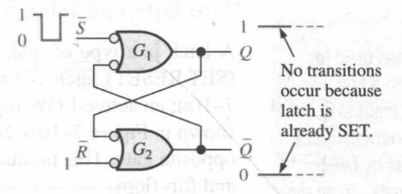
SET means that the Q output is HIGH.

Figure 7–3 illustrates the active-LOW input $\overline{S}$-$\overline{R}$ latch operation for each of the four possible combinations of levels on the inputs. (The first three combinations are valid, but the last is not.) Table 7–1 summarizes the logic operation in truth table form. Operation of the active-HIGH input NOR gate latch in Figure 7–1(a) is similar but requires the use of opposite logic levels.

RESET means that the Q output is LOW.

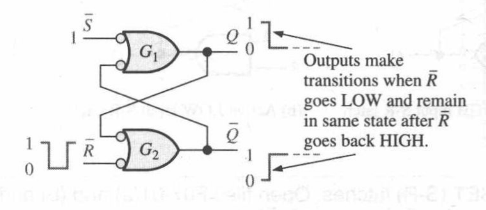


Latch starts out RESET (Q = 0).

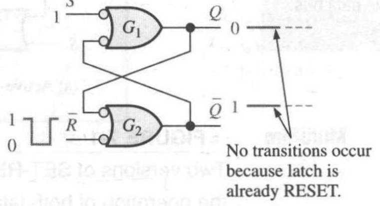


Latch starts out SET (Q = 1).

(a) Two possibilities for the SET operation

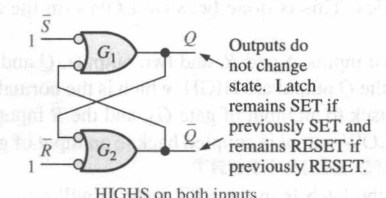


Latch starts out SET (Q = 1).

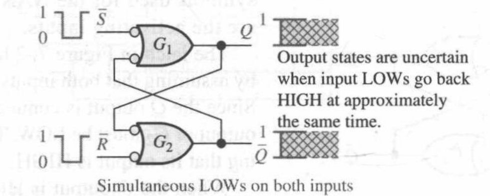


Latch starts out RESET (Q = 0).

(b) Two possibilities for the RESET operation



HIGHS on both inputs



Simultaneous LOWs on both inputs

(c) No-change condition

(d) Invalid condition

▲ **FIGURE 7-3**

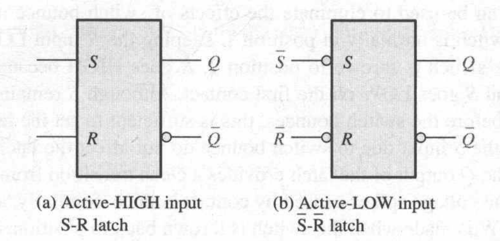
The three modes of basic $\overline{S}$-$\overline{R}$ latch operation (SET, RESET, no-change) and the invalid condition.

TABLE 7-1
Truth table for an active-LOW input $\overline{S}$-$\overline{R}$ latch.

Inputs		Outputs		
$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$	Comments
1	1	NC	NC	No change. Latch remains in present state.
0	1	1	0	Latch SET.
1	0	0	1	Latch RESET.
0	0	1	1	Invalid condition

Logic symbols for both the active-HIGH input and the active-LOW input latches are shown in Figure 7–4.

FIGURE 7-4
Logic symbols for the S-R and $\overline{S}$-$\overline{R}$ latch.



(a) Active-HIGH input S-R latch (b) Active-LOW input $\overline{S}$-$\overline{R}$ latch

Example 7–1 illustrates how an active-LOW input $\overline{S}$-$\overline{R}$ latch responds to conditions on its inputs. LOW levels are pulsed on each input in a certain sequence and the resulting Q output waveform is observed. The $\overline{S} = 0, \overline{R} = 0$ condition is avoided because it results in an invalid mode of operation and is a major drawback of any SET-RESET type of latch.

EXAMPLE 7-1

If the $\overline{S}$ and $\overline{R}$ waveforms in Figure 7–5(a) are applied to the inputs of the latch in Figure 7–4(b), determine the waveform that will be observed on the Q output. Assume that Q is initially LOW.

FIGURE 7-5

(a) $\overline{S}$, $\overline{R}$

(b) Q

Solution

See Figure 7–5(b).

Related Problem*

Determine the Q output of an active-HIGH input S-R latch if the waveforms in Figure 7–5(a) are inverted and applied to the inputs.

*Answers are at the end of the chapter.

An Application

The Latch as a Contact-Bounce Eliminator

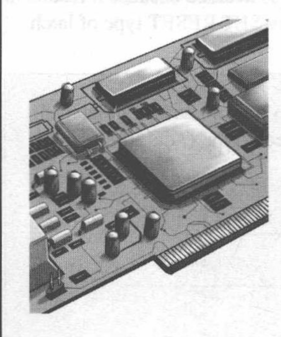
A good example of an application of an $\overline{S}$-$\overline{R}$ latch is in the elimination of mechanical switch contact "bounce." When the pole of a switch strikes the contact upon switch closure, it physically vibrates or bounces several times before finally making a solid contact. Although these bounces are very short in duration, they produce voltage spikes that are often not acceptable in a digital system. This situation is illustrated in Figure 7–6(a).

(a) Switch contact bounce (b) Contact-bounce eliminator circuit

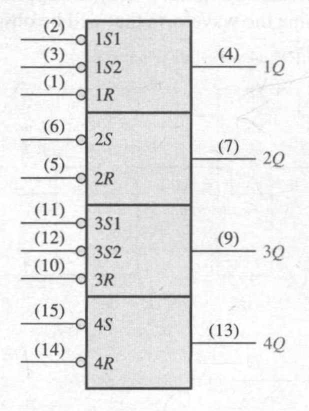
◀ **FIGURE 7-6**
The $\overline{S}$-$\overline{R}$ latch used to eliminate switch contact bounce.

An $\overline{S}$-$\overline{R}$ latch can be used to eliminate the effects of switch bounce as shown in Figure 7–6(b). The switch is normally in position 1, keeping the $\overline{R}$ input LOW and the latch RESET. When the switch is thrown to position 2, $\overline{R}$ goes HIGH because of the pull-up resistor to V_{CC}, and $\overline{S}$ goes LOW on the first contact. Although $\overline{S}$ remains LOW for only a very short time before the switch bounces, this is sufficient to set the latch. Any further voltage spikes on the $\overline{S}$ input due to switch bounce do not affect the latch, and it remains SET. Notice that the Q output of the latch provides a clean transition from LOW to HIGH, thus eliminating the voltage spikes caused by contact bounce. Similarly, a clean transition from HIGH to LOW is made when the switch is thrown back to position 1.

IMPLEMENTATION: $\overline{S}$-$\overline{R}$ LATCH



Fixed-Function Device The 74HC279A is a quad $\overline{S}$-$\overline{R}$ latch represented by the logic diagram of Figure 7–7(a) and the pin diagram in part (b). Notice that two of the latches each have two $\overline{S}$ inputs.



(a) Logic diagram

(b) Pin diagram

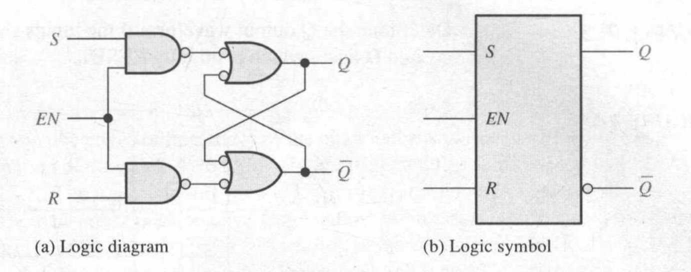
▲ **FIGURE 7-7**
The 74HC279A quad $\overline{S}$-$\overline{R}$ latch.

The Gated S-R Latch

A gated latch requires an enable input, *EN* (*G* is also used to designate an enable input). The logic diagram and logic symbol for a gated S-R latch are shown in Figure 7–8. The *S* and *R* inputs control the state to which the latch will go when a HIGH level is applied to the *EN* input. The latch will not change until *EN* is HIGH; but as long as it remains HIGH, the output is controlled by the state of the *S* and *R* inputs. The gated latch is a *level-sensitive* device. In this circuit, the invalid state occurs when both *S* and *R* are simultaneously HIGH and *EN* is also HIGH.

FIGURE 7-8

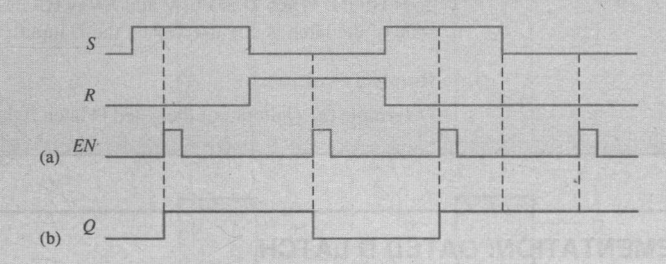
A gated S-R latch.



EXAMPLE 7-2

Determine the Q output waveform if the inputs shown in Figure 7–9(a) are applied to a gated S-R latch that is initially RESET.

FIGURE 7-9



Solution

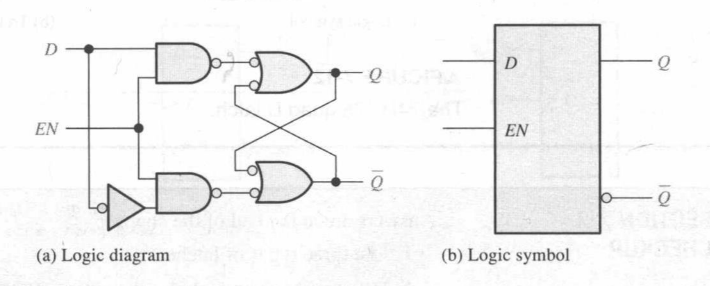
The Q waveform is shown in Figure 7–9(b). When S is HIGH and R is LOW, a HIGH on the EN input sets the latch. When S is LOW and R is HIGH, a HIGH on the EN input resets the latch. When both S and R are LOW, the Q output does not change from its present state.

Related Problem

Determine the Q output of a gated S-R latch if the S and R inputs in Figure 7–9(a) are inverted.

The Gated D Latch

Another type of gated latch is called the D latch. It differs from the S-R latch because it has only one input in addition to EN. This input is called the D (data) input. Figure 7–10 contains a logic diagram and logic symbol of a D latch. When the D input is HIGH and the EN input is HIGH, the latch will set. When the D input is LOW and EN is HIGH, the latch will reset. Stated another way, the output Q follows the input D when EN is HIGH.



MultiSim

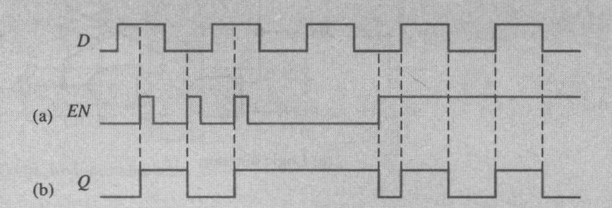
FIGURE 7-10

A gated D latch. Open file F07-10 and verify the operation.

EXAMPLE 7-3

Determine the Q output waveform if the inputs shown in Figure 7–11(a) are applied to a gated D latch, which is initially RESET.

▶ **FIGURE 7-11**



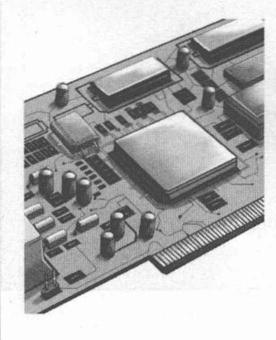
Solution

The Q waveform is shown in Figure 7–11(b). When D is HIGH and EN is HIGH, Q goes HIGH. When D is LOW and EN is HIGH, Q goes LOW. When EN is LOW, the state of the latch is not affected by the D input.

Related Problem

Determine the Q output of the gated D latch if the D input in Figure 7–11(a) is inverted.

IMPLEMENTATION: GATED D LATCH



Fixed-Function Device An example of a gated D latch is the 74HC75 represented by the logic symbol in Figure 7–12(a). The device has four latches. Notice that each active-HIGH EN input is shared by two latches and is designated as a control input (C). The truth table for each latch is shown in Figure 7–12(b). The X in the truth table represents a "don't care" condition. In this case, when the EN input is LOW, it does not matter what the D input is because the outputs are unaffected and remain in their prior states.

(a) Logic symbol

Inputs		Outputs		
D	EN	Q	$\overline{Q}$	Comments
0	1	0	1	RESET
1	1	1	0	SET
X	0	Q_0	$\overline{Q}_0$	No change

Note: Q_0 is the prior output level before the indicated input conditions were established.

(b) Truth table (each latch)

▲ **FIGURE 7-12**

The 74HC75 quad D latch.

SECTION 7-1 CHECKUP

Answers are at the end of the chapter.

1. List three types of latches.
2. Develop the truth table for the active-HIGH input S-R latch in Figure 7–1(a).
3. What is the Q output of a D latch when $EN = 1$ and $D = 1$?

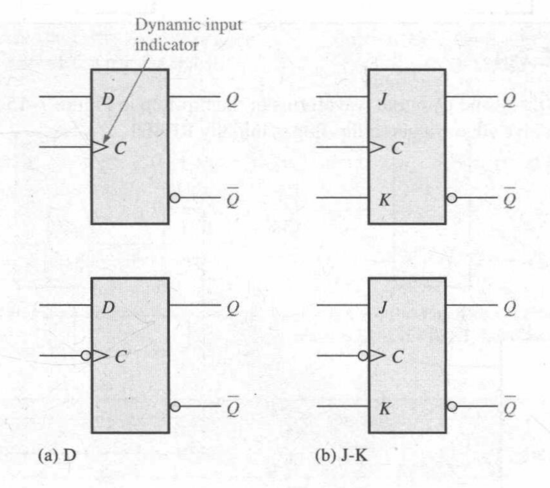
7-2 Flip-Flops

The dynamic input indicator ▷ means the flip-flop changes state only on the edge of a clock pulse.

An **edge-triggered flip-flop** changes state either at the positive edge (rising edge) or at the negative edge (falling edge) of the clock pulse and is sensitive to its inputs only at this transition of the clock. Two types of edge-triggered flip-flops are covered in this section: D and J-K. The logic symbols for these flip-flops are shown in Figure 7–13. Notice that each type can be either positive edge-triggered (no bubble at *C* input) or negative edge-triggered (bubble at *C* input). The key to identifying an edge-triggered flip-flop by its logic symbol is the small triangle inside the block at the clock (*C*) input. This triangle is called the *dynamic input indicator.*

▶ **FIGURE 7-13**

Edge-triggered flip-flop logic symbols (top: positive edge-triggered; bottom: negative edge-triggered).



The D Flip-Flop

D flip-flop but *D* as variable.

The *D* input of the **D flip-flop** is a **synchronous** input because data on the input are transferred to the flip-flop's output only on the triggering edge of the clock pulse. When *D* is HIGH, the *Q* output goes HIGH on the triggering edge of the clock pulse, and the flip-flop is SET. When *D* is LOW, the *Q* output goes LOW on the triggering edge of the clock pulse, and the flip-flop is RESET.

This basic operation of a positive edge-triggered D flip-flop is illustrated in Figure 7–14, and Table 7–2 is the truth table for this type of flip-flop. Remember, *the flip-flop cannot change state except on the triggering edge of a clock pulse.* The *D* input can be changed at any time when the clock input is LOW or HIGH (except for a very short interval around the triggering transition of the clock) without affecting the output. Just remember, *Q* follows *D* at the triggering edge of the clock.

InfoNote

Semiconductor memories consist of large numbers of individual cells. Each storage cell holds a 1 or a 0. One type of memory is the Static Random Access Memory or SRAM, which uses flip-flops for the storage cells because a flip-flop will retain either of its two states indefinitely as long as dc power is applied, thus the term *static.* This type of memory is classified as a *volatile* memory because all the stored data are lost when power is turned off. Another type of memory, the Dynamic Random Access Memory or DRAM, uses capacitance rather than flip-flops as the basic storage element and must be periodically refreshed in order to maintain the stored data.

(a) $D = 1$ flip-flop SETS on positive clock edge. (If already SET, it remains SET.)

(b) $D = 0$ flip-flop RESETS on positive clock edge. (If already RESET, it remains RESET.)

▲ **FIGURE 7-14**

Operation of a positive edge-triggered D flip-flop.

Inputs		Outputs		
D	CLK	Q	$\overline{Q}$	Comments
0	↑	0	1	RESET
1	↑	1	0	SET

↑ = clock transition LOW to HIGH

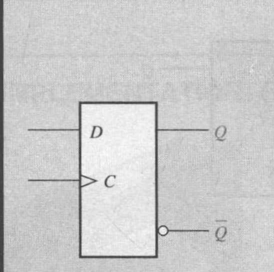
◀ **TABLE 7-2**

Truth table for a positive edge-triggered D flip-flop.

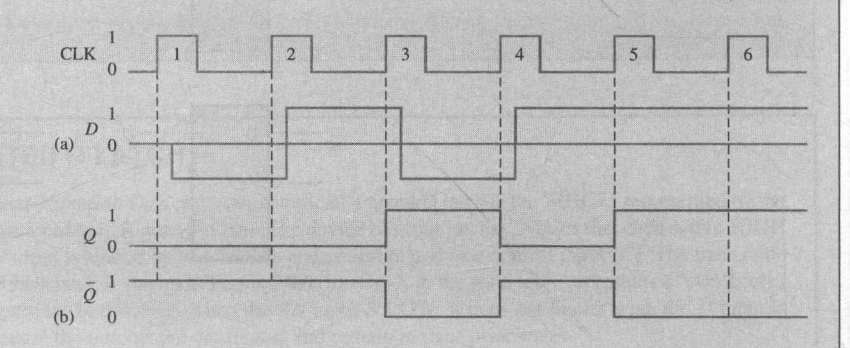
The operation and truth table for a negative edge-triggered D flip-flop are the same as those for a positive edge-triggered device except that the falling edge of the clock pulse is the triggering edge.

EXAMPLE 7-4

Determine the Q and $\overline{Q}$ output waveforms of the flip-flop in Figure 7–15 for the D and CLK inputs in Figure 7–16(a). Assume that the positive edge-triggered flip-flop is initially RESET.



▲ **FIGURE 7-15**



▲ **FIGURE 7-16**

Solution

1. At clock pulse 1, D is LOW, so Q remains LOW (RESET).
2. At clock pulse 2, D is LOW, so Q remains LOW (RESET).
3. At clock pulse 3, D is HIGH, so Q goes HIGH (SET).
4. At clock pulse 4, D is LOW, so Q goes LOW (RESET).
5. At clock pulse 5, D is HIGH, so Q goes HIGH (SET).
6. At clock pulse 6, D is HIGH, so Q remains HIGH (SET).

Once Q is determined, $\overline{Q}$ is easily found since it is simply the complement of Q. The resulting waveforms for Q and $\overline{Q}$ are shown in Figure 7–16(b) for the input waveforms in part (a).

Related Problem

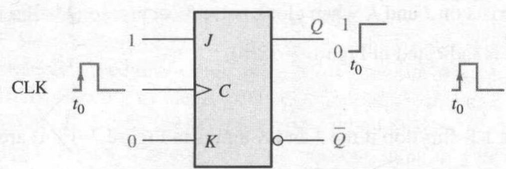
Determine Q and $\overline{Q}$ for the D input in Figure 7–16(a) if the flip-flop is a negative edge-triggered device.

The J-K Flip-Flop

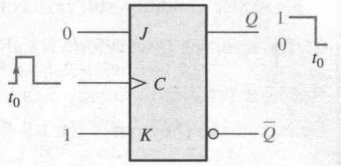
The J and K inputs of the **J-K flip-flop** are synchronous inputs because data on these inputs are transferred to the flip-flop's output only on the triggering edge of the clock pulse. When J is HIGH and K is LOW, the Q output goes HIGH on the triggering edge of the clock pulse, and the flip-flop is SET. When J is LOW and K is HIGH, the Q output goes LOW on the triggering edge of the clock pulse, and the flip-flop is RESET. When both J and K are LOW, the output does not change from its prior state. When J and K are both HIGH, the flip-flop changes state. This called the **toggle** mode.

This basic operation of a positive edge-triggered flip-flop is illustrated in Figure 7–17, and Table 7–3 is the truth table for this type of flip-flop. Remember, *the flip-flop cannot change state except on the triggering edge of a clock pulse.* The J and K inputs can be changed at any time when the clock input is LOW or HIGH (except for a very short interval around the triggering transition of the clock) without affecting the output.

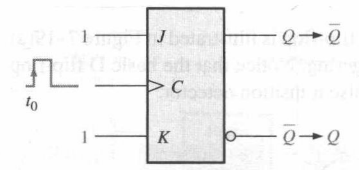
FIGURE 7-17
Operation of a positive edge-triggered J-K flip-flop.



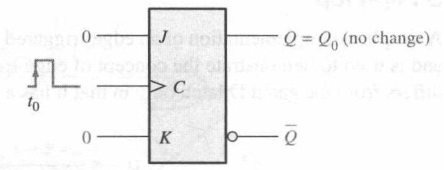
(a) $J = 1$, $K = 0$ flip-flop SETS on positive clock edge. (If already SET, it remains SET.)



(b) $J = 0$, $K = 1$ flip-flop RESETS on positive clock edge. (If already RESET, it remains RESET.)



(c) $J = 1$, $K = 1$ flip-flop changes state (toggle).



(d) $J = 0$, $K = 0$ flip-flop does not change. (If SET, it remains SET; if RESET, it remains RESET.)

TABLE 7-3
Truth table for a positive edge-triggered J-K flip-flop.

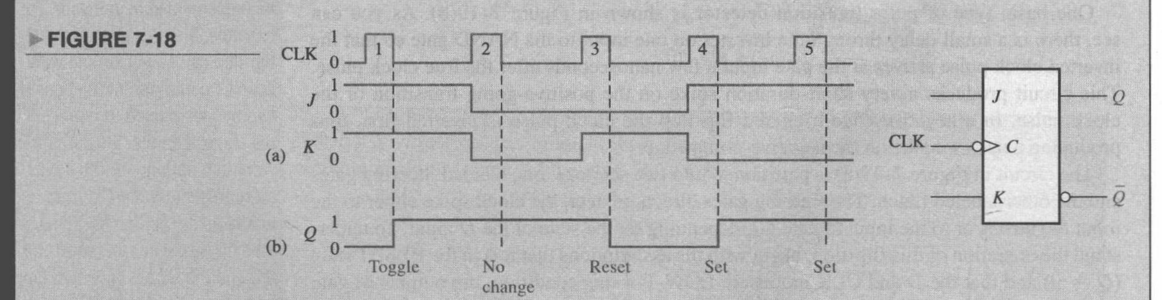
Inputs			Outputs		
J	K	CLK	Q	$\overline{Q}$	Comments
0	0	↑	Q_0	$\overline{Q}_0$	No change
0	1	↑	0	1	RESET
1	0	↑	1	0	SET
1	1	↑	$\overline{Q}_0$	Q_0	Toggle

↑ = clock transition LOW to HIGH
Q_0 = output level prior to clock transition

EXAMPLE 7-5

The waveforms in Figure 7–18(a) are applied to the J, K, and clock inputs as indicated. Determine the Q output, assuming that the flip-flop is initially RESET.

FIGURE 7-18



Solution

Since this is a negative edge-triggered flip-flop, as indicated by the "bubble" at the clock input, the Q output will change only on the negative-going edge of the clock pulse.

1. At the first clock pulse, both J and K are HIGH; and because this is a toggle condition, Q goes HIGH.
2. At clock pulse 2, a no-change condition exists on the inputs, keeping Q at a HIGH level.
3. When clock pulse 3 occurs, J is LOW and K is HIGH, resulting in a RESET condition; Q goes LOW.
4. At clock pulse 4, J is HIGH and K is LOW, resulting in a SET condition; Q goes HIGH.
5. A SET condition still exists on J and K when clock pulse 5 occurs, so Q will remain HIGH.

The resulting Q waveform is indicated in Figure 7–18(b).

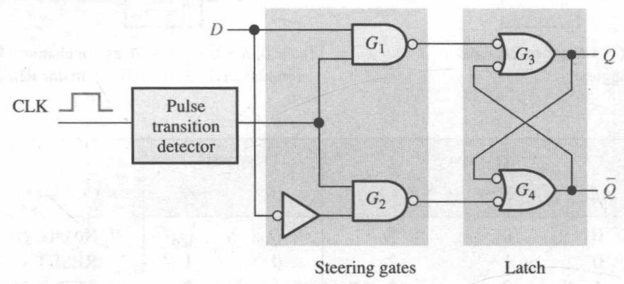
Related Problem

Determine the Q output of the J-K flip-flop if the J and K inputs in Figure 7–18(a) are inverted.

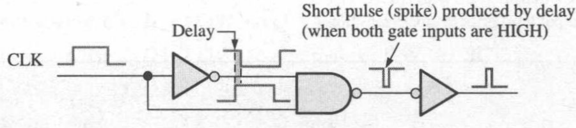
Edge-Triggered Operation

D Flip-Flop

A simplified implementation of an edge-triggered D flip-flop is illustrated in Figure 7–19(a) and is used to demonstrate the concept of edge-triggering. Notice that the basic D flip-flop differs from the gated D latch only in that it has a pulse transition detector.



(a) A simplified logic diagram for a positive edge-triggered D flip-flop



(b) A type of pulse transition detector

FIGURE 7-19
Edge triggering.

One basic type of pulse transition detector is shown in Figure 7–19(b). As you can see, there is a small delay through the inverter on one input to the NAND gate so that the inverted clock pulse arrives at the gate input a few nanoseconds after the true clock pulse. This circuit produces a very short-duration spike on the positive-going transition of the clock pulse. In a negative edge-triggered flip-flop the clock pulse is inverted first, thus producing a narrow spike on the negative-going edge.

The circuit in Figure 7–19(a) is partitioned into two sections, one labeled Steering gates and the other labeled Latch. The steering gates direct, or steer, the clock spike either to the input to gate G_3 or to the input to gate G_4, depending on the state of the D input. To understand the operation of this flip-flop, begin with the assumptions that it is in the RESET state ($Q = 0$) and that the D and CLK inputs are LOW. For this condition, the outputs of gate G_1 and gate G_2 are both HIGH. The LOW on the Q output is coupled back into one input of gate G_4, making the $\overline{Q}$ output HIGH. Because $\overline{Q}$ is HIGH, both inputs to gate G_3 are HIGH (remember, the output of gate G_1 is HIGH), holding the Q output LOW. If a pulse is applied to the CLK input, the outputs of gates G_1 and G_2 remain HIGH because they are disabled by the LOW on the D input; therefore, there is no change in the state of the flip-flop—it remains in the RESET state.

InfoNote

All logic operations that are performed with hardware can also be implemented in software. For example, the operation of a J-K flip-flop can be performed with specific computer instructions. If two bits were used to represent the J and K inputs, the computer would do nothing for 00, a data bit representing the Q output would be set (1) for 10, the Q data bit would be cleared (0) for 01, and the Q data bit would be complemented for 11. Although it may be unusual to use a computer to simulate a flip-flop, the point is that all hardware operations can be simulated using software.

The Q output of a D flip-flop assumes the state of the D input on the triggering edge of the clock.

Let's now make D HIGH and apply a clock pulse. Because the D input to gate G_1 is now HIGH, the output of gate G_1 goes LOW for a very short time (spike) when CLK goes HIGH, causing the Q output to go HIGH. Both inputs to gate G_4 are now HIGH (remember, gate G_2 output is HIGH because D is HIGH), forcing the $\overline{Q}$ output LOW. This LOW on $\overline{Q}$ is coupled back into one input of gate G_3, ensuring that the Q output will remain HIGH. The flip-flop is now in the SET state. Figure 7–20 illustrates the logic level transitions that take place within the flip-flop for this condition.

Next, let's make D LOW and apply a clock pulse. The positive-going edge of the clock produces a negative-going spike on the output of gate G_2, causing the $\overline{Q}$ output to go HIGH. Because of this HIGH on $\overline{Q}$, both inputs to gate G_3 are now HIGH (remember, the output of gate G_1 is HIGH because of the LOW on D), forcing the Q output to go LOW. This LOW on Q is coupled back into one input of gate G_4, ensuring that $\overline{Q}$ will remain HIGH. The flip-flop is now in the RESET state. Figure 7–21 illustrates the logic level transitions that occur within the flip-flop for this condition.

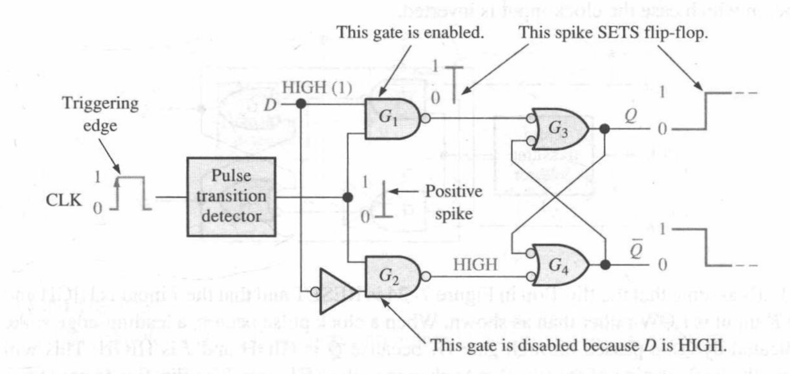


FIGURE 7-20 Flip-flop making a transition from the RESET state to the SET state on the positive-going edge of the clock pulse.

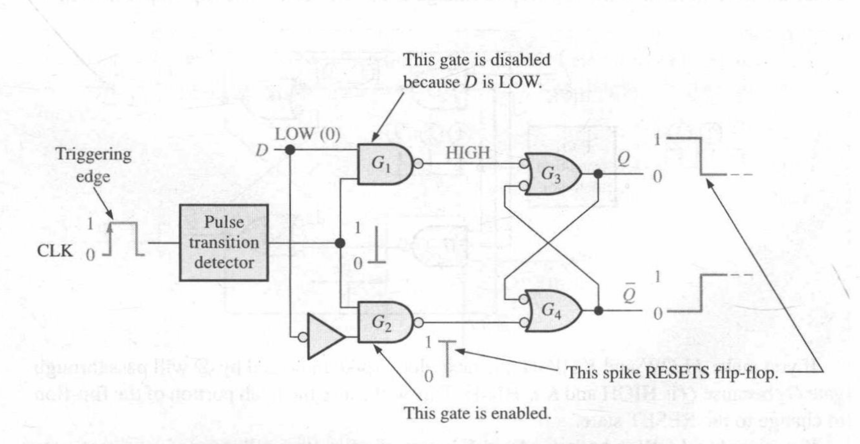
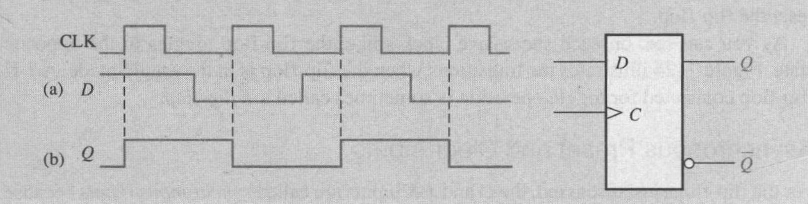


FIGURE 7-21 Flip-flop making a transition from the SET state to the RESET state on the positive-going edge of the clock pulse.

EXAMPLE 7-6

Given the waveforms in Figure 7–22(a) for the D input and the clock, determine the Q output waveform if the flip-flop starts out RESET.

FIGURE 7-22



Solution

The Q output goes to the state of the D input at the time of the positive-going clock edge. The resulting output is shown in Figure 7–22(b).

Related Problem

Determine the Q output for the D flip-flop if the D input in Figure 7–22(a) is inverted.

J-K Flip-Flop

Figure 7–23 shows the basic internal logic for a positive edge-triggered J-K flip-flop. The Q output is connected back to the input of gate G_2, and the $\overline{Q}$ output is connected back to the input of gate G_1. The two control inputs are labeled J and K in honor of Jack Kilby, who invented the integrated circuit. A J-K flip-flop can also be of the negative edge-triggered type, in which case the clock input is inverted.

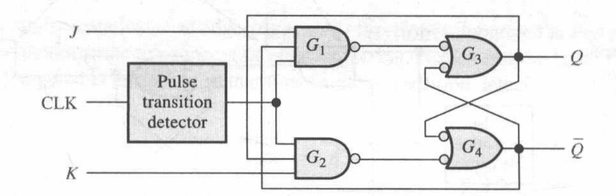


FIGURE 7-23 A simplified logic diagram for a positive edge-triggered J-K flip-flop.

Let's assume that the flip-flop in Figure 7–24 is RESET and that the J input is HIGH and the K input is LOW rather than as shown. When a clock pulse occurs, a leading-edge spike indicated by ① is passed through gate G_1 because $\overline{Q}$ is HIGH and J is HIGH. This will cause the latch portion of the flip-flop to change to the SET state. The flip-flop is now SET.

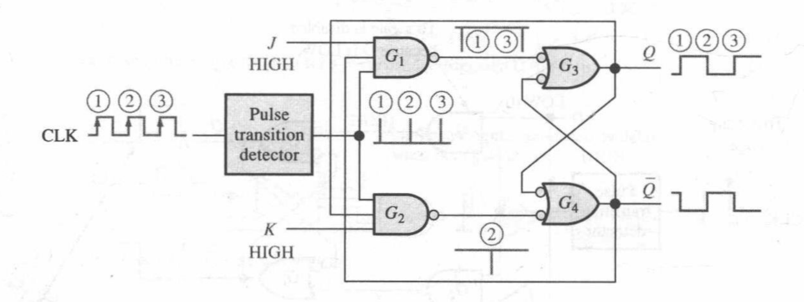


FIGURE 7-24 Transitions illustrating flip-flop operation.

If you make J LOW and K HIGH, the next clock spike indicated by ② will pass through gate G_2 because Q is HIGH and K is HIGH. This will cause the latch portion of the flip-flop to change to the RESET state.

In the toggle mode, a J-K flip-flop changes state on every clock pulse.

If you apply a LOW to both the J and K inputs, the flip-flop will stay in its present state when a clock pulse occurs. A LOW on both J and K results in a *no-change* condition.

When both the J and K inputs are HIGH and the flip-flop is RESET, the HIGH on the $\overline{Q}$ enables gate G_1; so the clock spike indicated by ③ passes through to set the flip-flop. Now there is a HIGH on Q, which allows the next clock spike to pass through gate G_2 and reset the flip-flop.

As you can see, on each successive clock spike, the flip-flop toggles to the opposite state. Figure 7–24 illustrates the transitions when the flip-flop is in the toggle mode. A J-K flip-flop connected for toggle operation is sometimes called a *T flip-flop.*

Asynchronous Preset and Clear Inputs

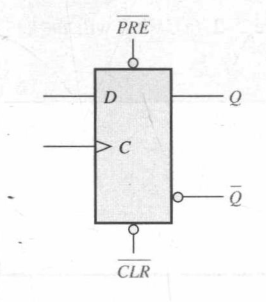
For the flip-flops just discussed, the D and J-K inputs are called *synchronous inputs* because data on these inputs are transferred to the flip-flop's output only on the triggering edge of the clock pulse; that is, the data are transferred synchronously with the clock.

An active preset input makes the Q output HIGH (SET).

An active clear input makes the Q output LOW (RESET).

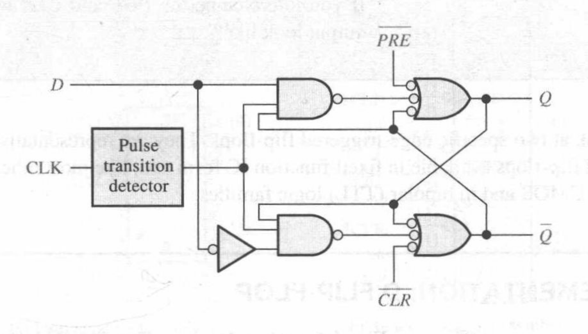
Most integrated circuit flip-flops also have **asynchronous** inputs. These are inputs that affect the state of the flip-flop *independent of the clock.* They are normally labeled **preset** (*PRE*) and **clear** (*CLR*), or *direct set* (S_D) and *direct reset* (R_D) by some manufacturers. An active level on the preset input will set the flip-flop, and an active level on the clear input will reset it. A logic symbol for a D flip-flop with preset and clear inputs is shown in Figure 7–25. These inputs are active-LOW, as indicated by the bubbles. These preset and clear inputs must both be kept HIGH for synchronous operation. In normal operation, preset and clear would not be LOW at the same time.

Figure 7–26 shows the logic diagram for an edge-triggered D flip-flop with active-LOW preset ($\overline{PRE}$) and clear ($\overline{CLR}$) inputs. This figure illustrates basically how these inputs work. As you can see, they are connected so that they override the effect of the synchronous input, D and the clock.



▲ **FIGURE 7-25**

Logic symbol for a D flip-flop with active-LOW preset and clear inputs.



▲ **FIGURE 7-26**

Logic diagram for a basic D flip-flop with active-LOW preset and clear inputs.

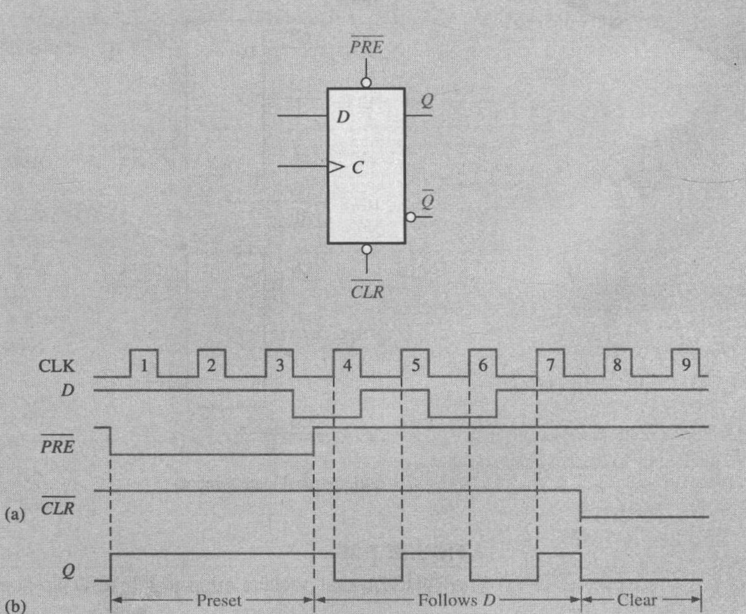
EXAMPLE 7-7

For the positive edge-triggered D flip-flop with preset and clear inputs in Figure 7–27, determine the Q output for the inputs shown in the timing diagram in part (a) if Q is initially LOW.

► **FIGURE 7-27**

Open file F07-27 to verify the operation.

MultiSim



Solution

1. During clock pulses 1, 2, and 3, the preset ($\overline{PRE}$) is LOW, keeping the flip-flop SET regardless of the synchronous D input.
2. For clock pulses 4, 5, 6, and 7, the output follows the input on the clock pulse because both $\overline{PRE}$ and $\overline{CLR}$ are HIGH.
3. For clock pulses 8 and 9, the clear ($\overline{CLR}$) input is LOW, keeping the flip-flop RESET regardless of the synchronous inputs.

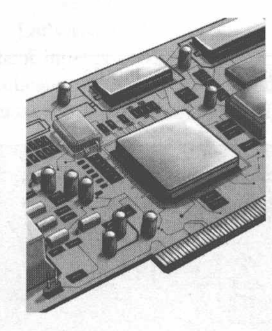
The resulting Q output is shown in Figure 7–27(b).

Related Problem

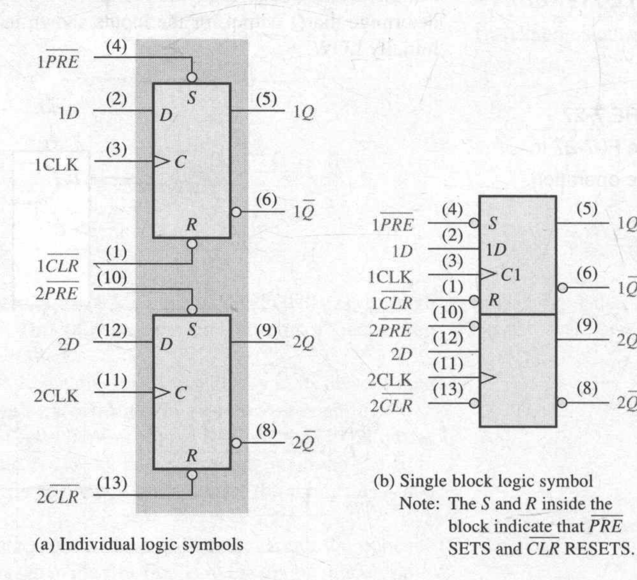
If you interchange the $\overline{PRE}$ and $\overline{CLR}$ waveforms in Figure 7–27(a), what will the Q output look like?

Let's look at two specific edge-triggered flip-flops. They are representative of the various types of flip-flops available in fixed-function IC form and, like most other devices, are available in CMOS and in bipolar (TTL) logic families.

IMPLEMENTATION: D FLIP-FLOP



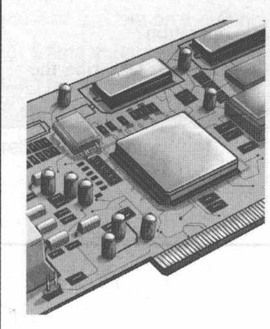
Fixed-Function Device The 74HC74 dual D flip-flop contains two identical D flip-flops that are independent of each other except for sharing V_{CC} and ground. The flip-flops are positive edge-triggered and have active-LOW asynchronous preset and clear inputs. The logic symbols for the individual flip-flops within the package are shown in Figure 7–28(a), and an ANSI/IEEE standard single block symbol that represents the entire device is shown in part (b). The pin numbers are shown in parentheses.



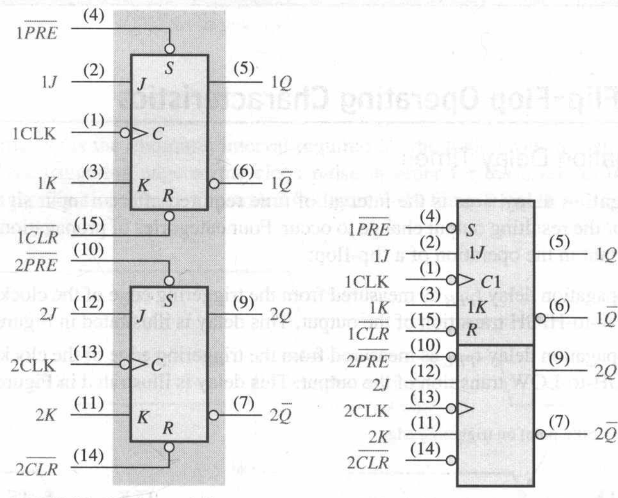
▲**FIGURE 7-28**

The 74HC74 dual positive edge-triggered D flip-flop.

MPLEMENTATION: J-K FLIP-FLOP



Fixed-Function Device The 74HC112 dual J-K flip-flop has two identical flip-flops that are negative edge-triggered and have active-LOW asynchronous preset and clear inputs. The logic symbols are shown in Figure 7–29.



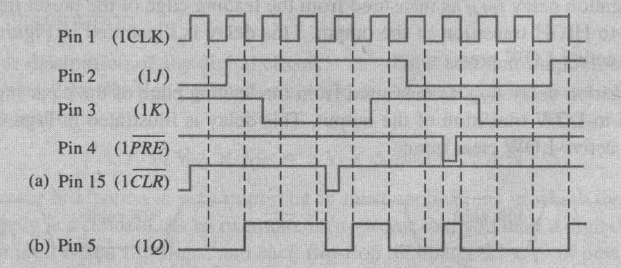
▲ **FIGURE 7-29**

The 74HC112 dual negative edge-triggered J-K flip-flop.

EXAMPLE 7-8

The $1J$, $1K$, 1CLK, $1\overline{PRE}$, and $1\overline{CLR}$ waveforms in Figure 7–30(a) are applied to one of the negative edge-triggered flip-flops in a 74HC112 package. Determine the $1Q$ output waveform.

▶ **FIGURE 7-30**



Solution

The resulting $1Q$ waveform is shown in Figure 7–30(b). Notice that each time a LOW is applied to the $1\overline{PRE}$ or $1\overline{CLR}$, the flip-flop is set or reset regardless of the states of the other inputs.

Related Problem

Determine the $1Q$ output waveform if the waveforms for $1\overline{PRE}$ and $1\overline{CLR}$ are interchanged.

SECTION 7-2 CHECKUP

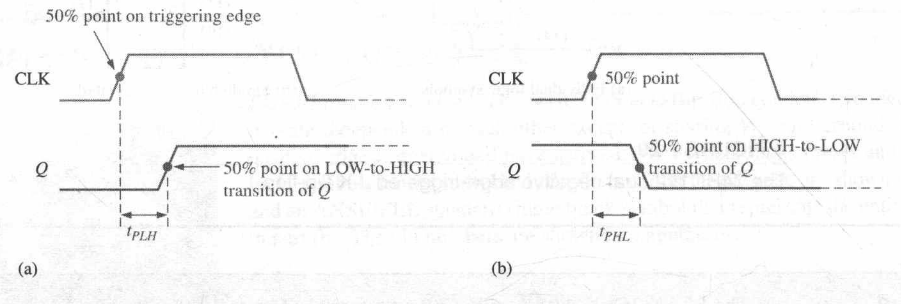
1. Describe the main difference between a gated D latch and an edge-triggered D flip-flop.
2. How does a J-K flip-flop differ from a D flip-flop in its basic operation?
3. Assume that the flip-flop in Figure 7–22 is negative edge-triggered. Describe the output waveform for the same CLK and *D* waveforms.

7-3 Flip-Flop Operating Characteristics

Propagation Delay Times

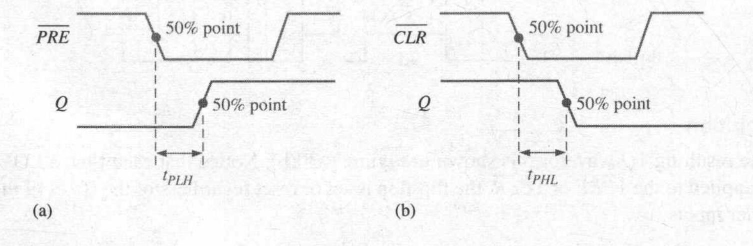
A **propagation delay time** is the interval of time required after an input signal has been applied for the resulting output change to occur. Four categories of propagation delay times are important in the operation of a flip-flop:

1. Propagation delay t_{PLH} as measured from the triggering edge of the clock pulse to the LOW-to-HIGH transition of the output. This delay is illustrated in Figure 7–31(a).
2. Propagation delay t_{PHL} as measured from the triggering edge of the clock pulse to the HIGH-to-LOW transition of the output. This delay is illustrated in Figure 7–31(b).



▲ **FIGURE 7-31**
Propagation delays, clock to output.

3. Propagation delay t_{PLH} as measured from the leading edge of the preset input to the LOW-to-HIGH transition of the output. This delay is illustrated in Figure 7–32(a) for an active-LOW preset input.
4. Propagation delay t_{PHL} as measured from the leading edge of the clear input to the HIGH-to-LOW transition of the output. This delay is illustrated in Figure 7–32(b) for an active-LOW clear input.



◀ **FIGURE 7-32**
Propagation delays, preset input to output and clear input to output.

Set-up Time

The **set-up time** (t_s) is the minimum interval required for the logic levels to be maintained constantly on the inputs (*J* and *K*, or *D*) prior to the triggering edge of the clock pulse in order for the levels to be reliably clocked into the flip-flop. This interval is illustrated in Figure 7–33 for a D flip-flop.

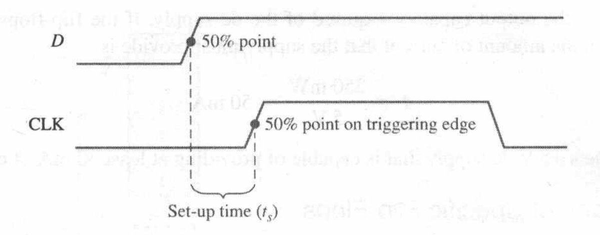


FIGURE 7-33

Set-up time (t_s). The logic level must be present on the *D* input for a time equal to or greater than t_s before the triggering edge of the clock pulse for reliable data entry.

Hold Time

The **hold time** (t_h) is the minimum interval required for the logic levels to remain on the inputs after the triggering edge of the clock pulse in order for the levels to be reliably clocked into the flip-flop. This is illustrated in Figure 7–34 for a D flip-flop.

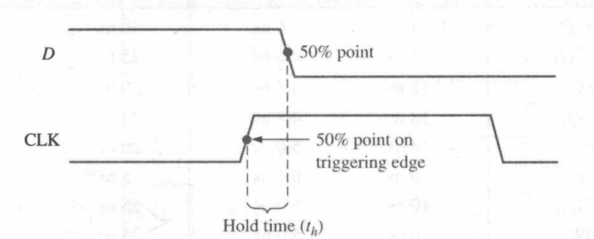


FIGURE 7-34

Hold time (t_h). The logic level must remain on the *D* input for a time equal to or greater than t_h after the triggering edge of the clock pulse for reliable data entry.

Maximum Clock Frequency

The maximum clock frequency (f_{max}) is the highest rate at which a flip-flop can be reliably triggered. At clock frequencies above the maximum, the flip-flop would be unable to respond quickly enough, and its operation would be impaired.

Pulse Widths

Minimum pulse widths (t_W) for reliable operation are usually specified by the manufacturer for the clock, preset, and clear inputs. Typically, the clock is specified by its minimum HIGH time and its minimum LOW time.

Power Dissipation

The **power dissipation** of any digital circuit is the total power consumption of the device. For example, if the flip-flop operates on a +5 V dc source and draws 5 mA of current, the power dissipation is

$$P = V_{CC} \times I_{CC} = 5\text{ V} \times 5\text{ mA} = 25\text{ mW}$$

The power dissipation is very important in most applications in which the capacity of the dc supply is a concern. As an example, let's assume that you have a digital system that requires a total of ten flip-flops, and each flip-flop dissipates 25 mW of power. The total power requirement is

$$P_T = 10 \times 25\text{ mW} = 250\text{ mW} = 0.25\text{ W}$$

An advantage of CMOS is that it can operate over a wider range of dc supply voltages (typically 2 V to 6 V) than bipolar and, therefore, less expensive power supplies that do not have precise regulation can be used. Also, batteries can be used as secondary or primary sources for CMOS circuits. In addition, lower voltages mean that the IC dissipates less power. The drawback is that the performance of CMOS is degraded with lower supply voltages. For example, the guaranteed maximum clock frequency of a CMOS flip-flop is much less at $V_{CC} = 2$ V than at $V_{CC} = 6$ V.

This tells you the output capacity required of the dc supply. If the flip-flops operate on +5 V dc, then the amount of current that the supply must provide is

$$I = \frac{250\ \text{mW}}{5\ \text{V}} = 50\ \text{mA}$$

You must use a +5 V dc supply that is capable of providing at least 50 mA of current.

Comparison of Specific Flip-Flops

Table 7–4 provides a comparison, in terms of the operating parameters discussed in this section, of four CMOS and bipolar (TTL) flip-flops of the same type but with different IC families (HC, AHC, LS, and F).

TABLE 7-4
Comparison of operating parameters for four IC families of flip-flops of the same type at 25°C.

	CMOS		Bipolar (TTL)	
Parameter	**74HC74A**	**74AHC74**	**74LS74A**	**74F74**
t_{PHL} (CLK to Q)	17 ns	4.6 ns	40 ns	6.8 ns
t_{PLH} (CLK to Q)	17 ns	4.6 ns	25 ns	8.0 ns
t_{PHL}($\overline{CLR}$ to Q)	18 ns	4.8 ns	40 ns	9.0 ns
t_{PLH}($\overline{PRE}$ to Q)	18 ns	4.8 ns	25 ns	6.1 ns
t_s (set-up time)	14 ns	5.0 ns	20 ns	2.0 ns
t_h (hold time)	3.0 ns	0.5 ns	5 ns	1.0 ns
t_W (CLK HIGH)	10 ns	5.0 ns	25 ns	4.0 ns
t_W (CLK LOW)	10 ns	5.0 ns	25 ns	5.0 ns
t_W($\overline{CLR}/\overline{PRE}$)	10 ns	5.0 ns	25 ns	4.0 ns
f_{max}	35 MHz	170 MHz	25 MHz	100 MHz
Power, quiescent	0.012 mW	1.1 mW		
Power, 50% duty cycle			44 mW	88 mW

SECTION 7-3 CHECKUP

1. Define the following:
   (a) set-up time (b) hold time
2. Which specific flip-flop in Table 7–4 can be operated at the highest frequency?

7-4 Flip-Flop Applications

Parallel Data Storage

A common requirement in digital systems is to store several bits of data from parallel lines simultaneously in a group of flip-flops. This operation is illustrated in Figure 7–35(a) using four flip-flops. Each of the four parallel data lines is connected to the D input of a flip-flop. The clock inputs of the flip-flops are connected together, so that each flip-flop is triggered by the same clock pulse. In this example, positive edge-triggered flip-flops are used, so the data on the D inputs are stored simultaneously by the flip-flops on the positive edge of the clock, as indicated in the timing diagram in Figure 7–35(b). Also, the asynchronous reset (R) inputs are connected to a common $\overline{CLR}$ line, which initially resets all the flip-flops.

This group of four flip-flops is an example of a basic register used for data storage. In digital systems, data are normally stored in groups of bits (usually eight or multiples thereof) that represent numbers, codes, or other information. Registers are covered in Chapter 8.

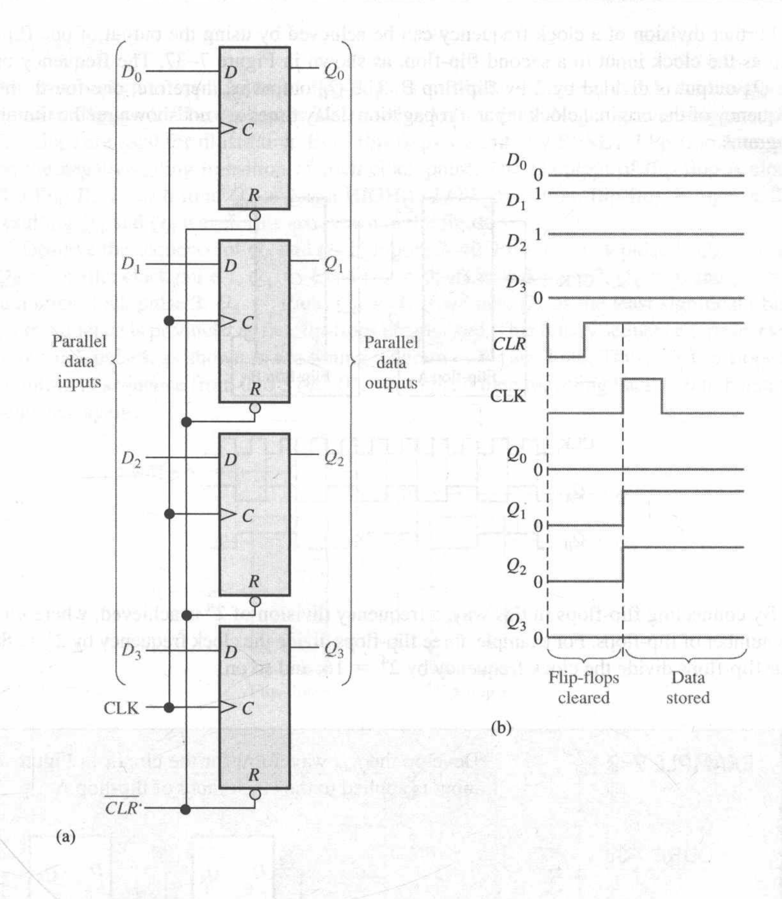


FIGURE 7-35 Example of flip-flops used in a basic register for parallel data storage.

Frequency Division

Another application of a flip-flop is dividing (reducing) the frequency of a periodic waveform. When a pulse waveform is applied to the clock input of a D or J-K flip-flop that is connected to toggle ($D = \overline{Q}$ or $J = K = 1$), the Q output is a square wave with one-half the frequency of the clock input. Thus, a single flip-flop can be applied as a divide-by-2 device, as is illustrated in Figure 7–36 for both a D and a J-K flip-flop. As you can see in part (c), the flip-flop changes state on each triggering clock edge (positive edge-triggered in this case). This results in an output that changes at half the frequency of the clock waveform.

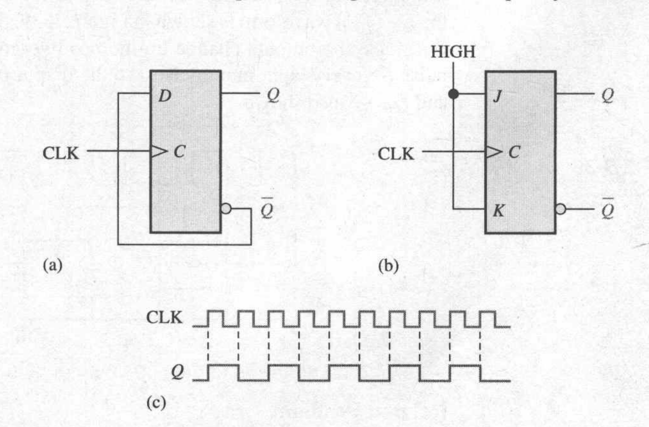
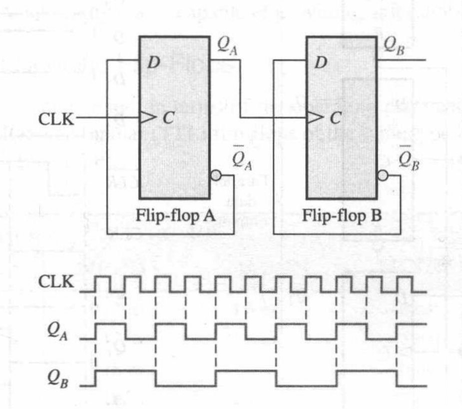


FIGURE 7-36 The D flip-flop and J-K flip-flop as a divide-by-2 device. Q is one-half the frequency of CLK. Open file F07-36 and verify the operation.

MultiSim

Further division of a clock frequency can be achieved by using the output of one flip-flop as the clock input to a second flip-flop, as shown in Figure 7–37. The frequency of the Q_A output is divided by 2 by flip-flop B. The Q_B output is, therefore, one-fourth the frequency of the original clock input. Propagation delay times are not shown on the timing diagrams.



◀ FIGURE 7-37
Example of two D flip-flops used to divide the clock frequency by 4. Q_A is one-half and Q_B is one-fourth the frequency of CLK. Open file F07-37 and verify the operation.

MultiSim

By connecting flip-flops in this way, a frequency division of 2^n is achieved, where n is the number of flip-flops. For example, three flip-flops divide the clock frequency by $2^3 = 8$; four flip-flops divide the clock frequency by $2^4 = 16$; and so on.

EXAMPLE 7-9

Develop the f_{out} waveform for the circuit in Figure 7–38 when an 8 kHz square wave input is applied to the clock input of flip-flop A.

▶ FIGURE 7-38

Solution

The three flip-flops are connected to divide the input frequency by eight ($2^3 = 8$) and the Q_C (f_{out}) waveform is shown in Figure 7–39. Since these are positive edge-triggered flip-flops, the outputs change on the positive-going clock edge. There is one output pulse for every eight input pulses, so the output frequency is 1 kHz. Waveforms of Q_A and Q_B are also shown.

▶ FIGURE 7-39

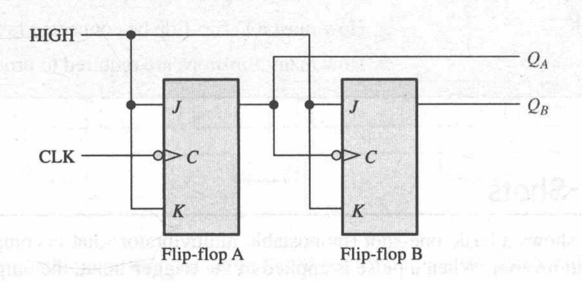
Related Problem

How many flip-flops are required to divide a frequency by thirty-two?

Counting

Another important application of flip-flops is in digital counters, which are covered in detail in Chapter 9. The concept is illustrated in Figure 7–40. Negative edge-triggered J-K flip-flops are used for illustration. Both flip-flops are initially RESET. Flip-flop A toggles on the negative-going transition of each clock pulse. The Q output of flip-flop A clocks flip-flop B, so each time Q_A makes a HIGH-to-LOW transition, flip-flop B toggles. The resulting Q_A and Q_B waveforms are shown in the figure.

Observe the sequence of Q_A and Q_B in Figure 7–40. Prior to clock pulse 1, $Q_A = 0$ and $Q_B = 0$; after clock pulse 1, $Q_A = 1$ and $Q_B = 0$; after clock pulse 2, $Q_A = 0$ and $Q_B = 1$; and after clock pulse 3, $Q_A = 1$ and $Q_B = 1$. If we take Q_A as the least significant bit, a 2-bit sequence is produced as the flip-flops are clocked. This binary sequence repeats every four clock pulses, as shown in the timing diagram of Figure 7–40. Thus, the flip-flops are counting in sequence from 0 to 3 (00, 01, 10, 11) and then recycling back to 0 to begin the sequence again.



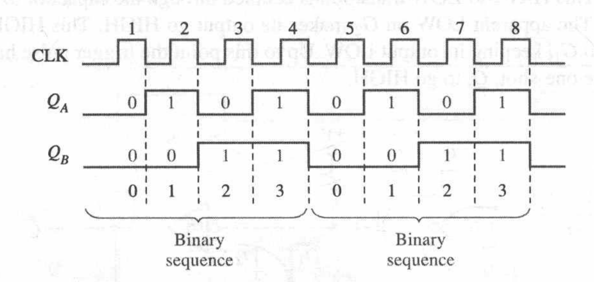
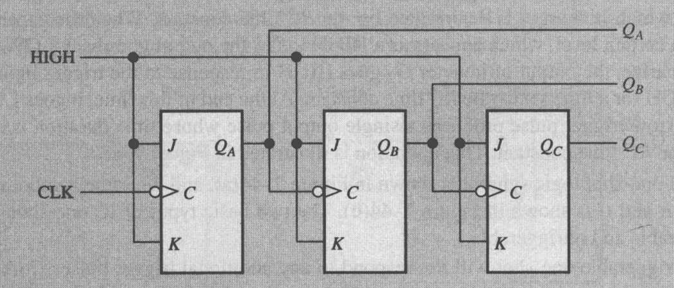


FIGURE 7-40 J-K flip-flops used to generate a binary count sequence (00, 01, 10, 11). Two repetitions are shown.

EXAMPLE 7-10

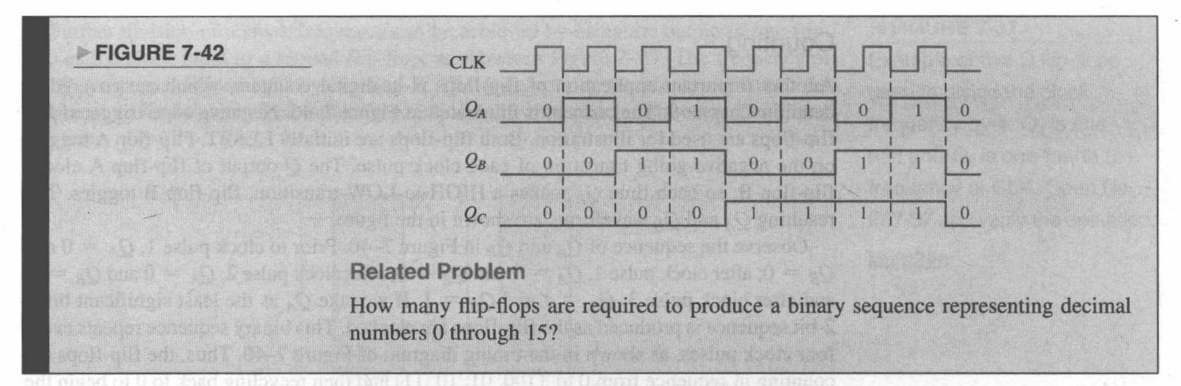
Determine the output waveforms in relation to the clock for Q_A, Q_B, and Q_C in the circuit of Figure 7–41 and show the binary sequence represented by these waveforms.

FIGURE 7-41



Solution

The output timing diagram is shown in Figure 7–42. Notice that the outputs change on the negative-going edge of the clock pulses. The outputs go through the binary sequence 000, 001, 010, 011, 100, 101, 110, and 111 as indicated.



▶ FIGURE 7-42

Related Problem

How many flip-flops are required to produce a binary sequence representing decimal numbers 0 through 15?

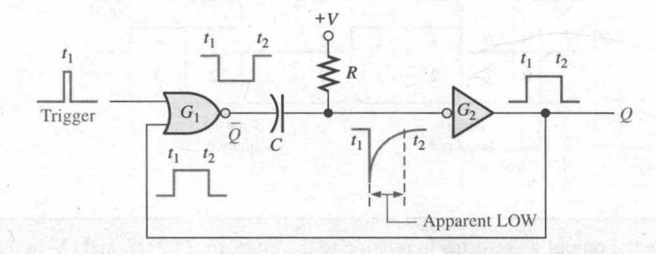
SECTION 7-4 CHECKUP

1. What is a group of flip-flops used for data storage called?
2. How must a D flip-flop be connected to function as a divide-by-2 device?
3. How many flip-flops are required to produce a divide-by-64 device?

7-5 One-Shots

A one-shot produces a single pulse each time it is triggered.

Figure 7–43 shows a basic one-shot (monostable multivibrator) that is composed of a logic gate and an inverter. When a pulse is applied to the **trigger** input, the output of gate G_1 goes LOW. This HIGH-to-LOW transition is coupled through the capacitor to the input of inverter G_2. The apparent LOW on G_2 makes its output go HIGH. This HIGH is connected back into G_1, keeping its output LOW. Up to this point the trigger pulse has caused the output of the one-shot, Q, to go HIGH.



◀ FIGURE 7-43

A simple one-shot circuit.

The capacitor immediately begins to charge through R toward the high voltage level. The rate at which it charges is determined by the RC time constant. When the capacitor charges to a certain level, which appears as a HIGH to G_2, the output goes back LOW.

To summarize, the output of inverter G_2 goes HIGH in response to the trigger input. It remains HIGH for a time set by the RC time constant. At the end of this time, it goes LOW. A single narrow trigger pulse produces a single output pulse whose time duration is controlled by the RC time constant. This operation is illustrated in Figure 7–43.

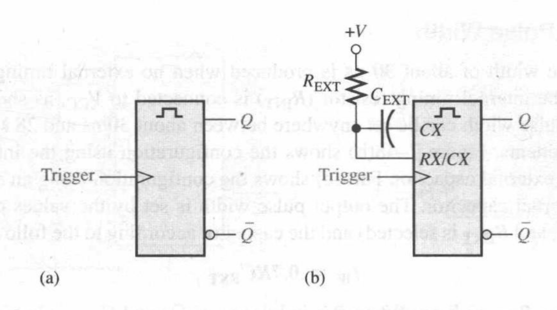
A typical one-shot logic symbol is shown in Figure 7–44(a), and the same symbol with an external R and C is shown in Figure 7–44(b). The two basic types of IC one-shots are nonretriggerable and retriggerable.

A nonretriggerable one-shot will not respond to any additional trigger pulses from the time it is triggered into its unstable state until it returns to its stable state. In other words, it will ignore any trigger pulses occurring before it times out. The time that the one-shot remains in its unstable state is the pulse width of the output.

Figure 7–45 shows the nonretriggerable one-shot being triggered at intervals greater than its pulse width and at intervals less than the pulse width. Notice that in the second case, the additional pulses are ignored.

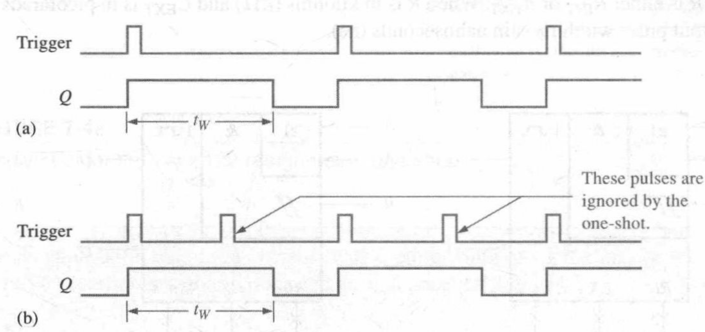
▶ FIGURE 7-44

Basic one-shot logic symbols. *CX* and *RX* stand for external components.



▶ FIGURE 7-45

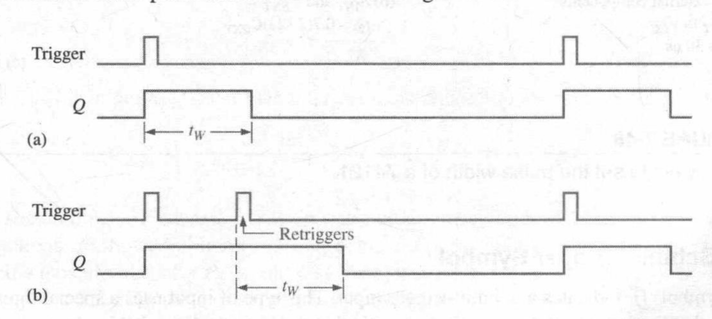
Nonretriggerable one-shot action.



A retriggerable one-shot can be triggered before it times out. The result of retriggering is an extension of the pulse width as illustrated in Figure 7–46.

▶ FIGURE 7-46

Retriggerable one-shot action.

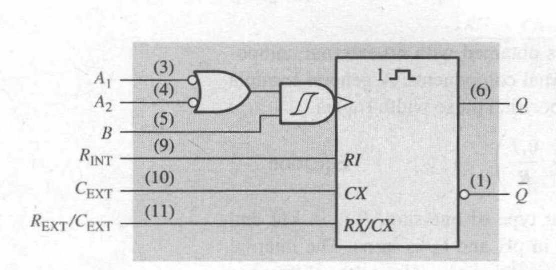


Nonretriggerable One-Shot

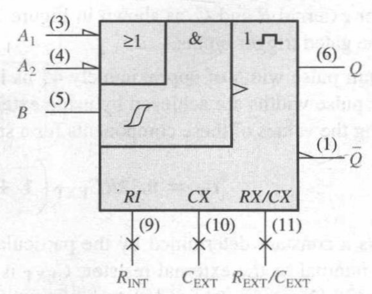
The 74121 is an example of a nonretriggerable IC one-shot. It has provisions for external *R* and *C*, as shown in Figure 7–47. The inputs labeled A_1, A_2, and *B* are gated trigger inputs. The R_{INT} input connects to a 2 kΩ internal timing resistor.

▼ FIGURE 7-47

Logic symbols for the 74121 nonretriggerable one-shot.



(a) Traditional logic symbol



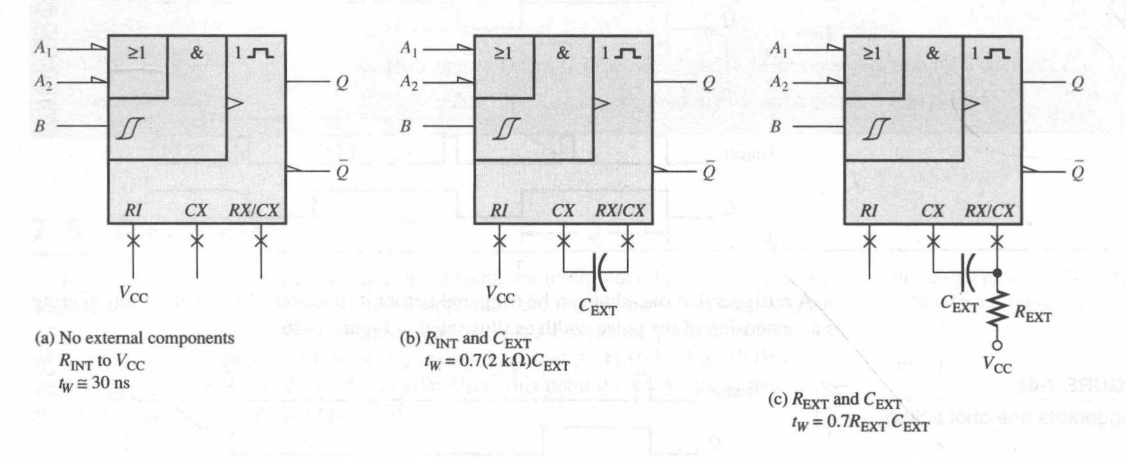
(b) ANSI/IEEE std. 91–1984 logic symbol (× = nonlogic connection). "1⎍" is the qualifying symbol for a nonretriggerable one-shot.

Setting the Pulse Width

A typical pulse width of about 30 ns is produced when no external timing components are used and the internal timing resistor (R_{INT}) is connected to V_{CC}, as shown in Figure 7–48(a). The pulse width can be set anywhere between about 30 ns and 28 s by the use of external components. Figure 7–48(b) shows the configuration using the internal resistor (2 kΩ) and an external capacitor. Part (c) shows the configuration using an external resistor and an external capacitor. The output pulse width is set by the values of the resistor (R_{INT} = 2 kΩ, and R_{EXT} is selected) and the capacitor according to the following formula:

$$t_W = 0.7RC_{EXT} \qquad \text{Equation 7–1}$$

where R is either R_{INT} or R_{EXT}. When R is in kilohms (kΩ) and C_{EXT} is in picofarads (pF), the output pulse width t_W is in nanoseconds (ns).



▲ **FIGURE 7-48**

Three ways to set the pulse width of a 74121.

The Schmitt-Trigger Symbol

The symbol ∫ indicates a Schmitt-trigger input. This type of input uses a special threshold circuit that produces **hysteresis**, a characteristic that prevents erratic switching between states when a slow-changing trigger voltage hovers around the critical input level. This allows reliable triggering to occur even when the input is changing as slowly as 1 volt/second.

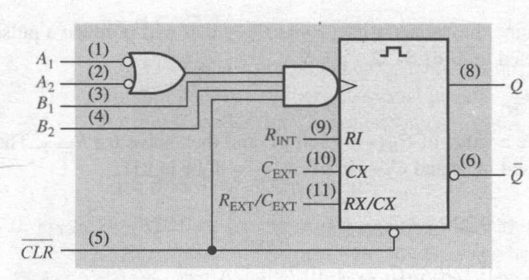
Retriggerable One-Shot

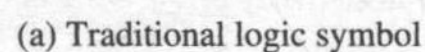
The 74LS122 is an example of a retriggerable IC one-shot with a clear input. It also has provisions for external R and C, as shown in Figure 7–49. The inputs labeled A_1, A_2, B_1, and B_2 are the gated trigger inputs.

A minimum pulse width of approximately 45 ns is obtained with no external components. Wider pulse widths are achieved by using external components. A general formula for calculating the values of these components for a specified pulse width (t_W) is

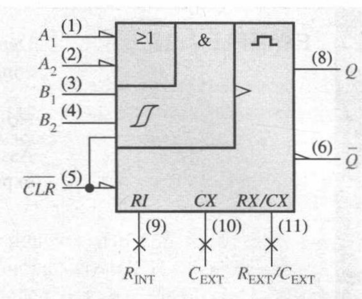
$$t_W = 0.32RC_{EXT}\left(1 + \frac{0.7}{R}\right) \qquad \text{Equation 7–2}$$

where 0.32 is a constant determined by the particular type of one-shot, R is in kΩ and is either the internal or the external resistor, C_{EXT} is in pF, and t_W is in ns. The internal resistance is 10 kΩ and can be used instead of an external resistor. (Notice the difference between this formula and that for the 74121, shown in Equation 7–1.)





(a) Traditional logic symbol



(b) ANSI/IEEE std. 91–1984 logic symbol (X = nonlogic connection). ⎍ is the qualifying symbol for a retriggerable one-shot.

▲FIGURE 7-49
Logic symbol for the 74LS122 retriggerable one-shot.

EXAMPLE 7-11

A certain application requires a one-shot with a pulse width of approximately 100 ms. Using a 74121, show the connections and the component values.

Solution

Arbitrarily select R_{EXT} = **39 kΩ** and calculate the necessary capacitance.

$$t_W = 0.7R_{EXT}C_{EXT}$$

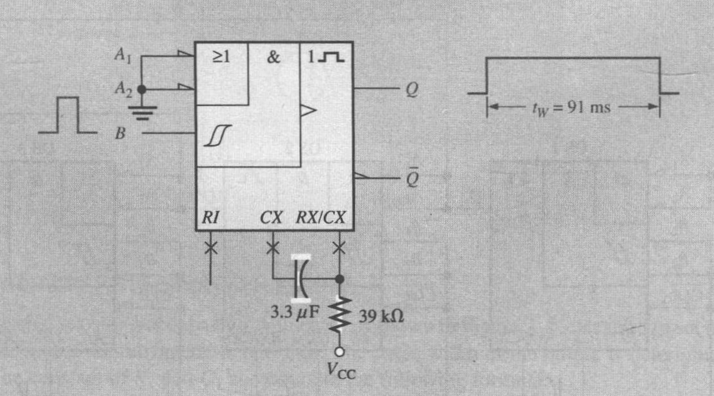
$$C_{EXT} = \frac{t_W}{0.7R_{EXT}}$$

where C_{EXT} is in pF, R_{EXT} is in kΩ, and t_W is in ns. Since 100 ms = 1×10^8 ns,

$$C_{EXT} = \frac{1 \times 10^8 \text{ ns}}{0.7(39 \text{ k}\Omega)} = 3.66 \times 10^{-6}\text{ pF} = \mathbf{3.66\ \mu F}$$

A standard 3.3 μF capacitor will give an output pulse width of 91 ms. The proper connections are shown in Figure 7–50. To achieve a pulse width closer to 100 ms, other combinations of values for R_{EXT} and C_{EXT} can be tried. For example, R_{EXT} = 68 kΩ and C_{EXT} = 2.2 μF gives a pulse width of 105 ms.

▶FIGURE 7-50



Related Problem

Use an external capacitor in conjunction with R_{INT} to produce an output pulse width of 10 μs from the 74121.

EXAMPLE 7-12

Determine the values of R_{EXT} and C_{EXT} that will produce a pulse width of 1 μs when connected to a 74LS122.

Solution

Assume a value of C_{EXT} = **560 pF** and then solve for R_{EXT}. The pulse width must be expressed in ns and C_{EXT} in pF. R_{EXT} will be in kΩ.

$$t_w = 0.32R_{EXT}C_{EXT}\left(1 + \frac{0.7}{R_{EXT}}\right) = 0.32R_{EXT}C_{EXT} + 0.7\left(\frac{0.32R_{EXT}C_{EXT}}{R_{EXT}}\right)$$

$$= 0.32R_{EXT}C_{EXT} + (0.7)(0.32)C_{EXT}$$

$$R_{EXT} = \frac{t_W - (0.7)(0.32)C_{EXT}}{0.32C_{EXT}} = \frac{t_W}{0.32C_{EXT}} - 0.7$$

$$= \frac{1000\text{ ns}}{(0.32)560\text{ pF}} - 0.7 = \mathbf{4.88\ k\Omega}$$

Use a standard value of **4.7 kΩ.**

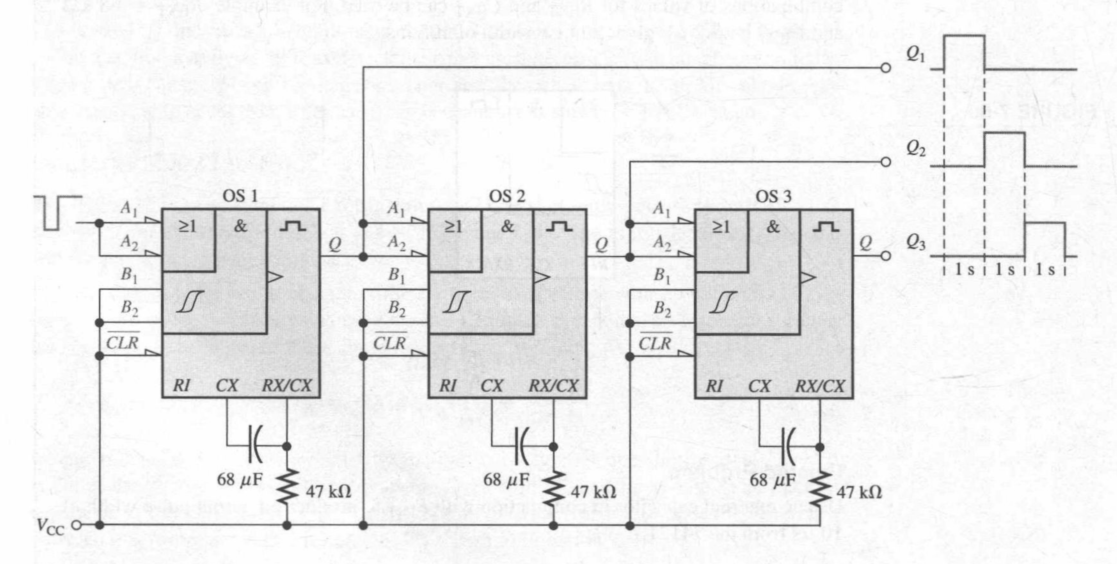
Related Problem

Show the connections and component values for a 74LS122 one-shot with an output pulse width of 5 μs. Assume C_{EXT} = 560 pF.

An Application

One practical one-shot application is a sequential timer that can be used to illuminate a series of lights. This type of circuit can be used, for example, in a lane change directional indicator for highway construction projects or in sequential turn signals on automobiles.

Figure 7–51 shows three 74LS122 one-shots connected as a sequential timer. This particular circuit produces a sequence of three 1 s pulses. The first one-shot is triggered by a switch closure or a low-frequency pulse input, producing a 1 s output pulse. When the first one-shot (OS 1) times out and the 1 s pulse goes LOW, the second one-shot (OS 2) is triggered, also producing a 1 s output pulse. When this second pulse goes LOW, the third one-shot (OS 3) is triggered and the third 1 s pulse is produced. The output timing is illustrated in the figure. Variations of this basic arrangement can be used to produce a variety of timed outputs.



▲ **FIGURE 7-51**

A sequential timing circuit using three 74LS122 one-shots.

The 555 Timer as a One-Shot

The 555 **timer** is a versatile and widely used IC device because it can be configured in two different modes as either a monostable multivibrator (one-shot) or as an astable multivibrator (pulse oscillator). The astable multivibrator is discussed in Section 7–6.

The 555 Timer Operation

A functional diagram showing the internal components of a 555 timer is shown in Figure 7–52. The comparators are devices whose outputs are HIGH when the voltage on the positive (+) input is greater than the voltage on the negative (−) input and LOW when the − input voltage is greater than the + input voltage. The voltage divider consisting of three 5 kΩ resistors provides a trigger level of $\frac{1}{3}V_{CC}$ and a threshold level of $\frac{2}{3}V_{CC}$. The control voltage input (pin 5) can be used to externally adjust the trigger and threshold levels to other values if necessary. When the normally HIGH trigger input momentarily goes below $\frac{1}{3}V_{CC}$, the output of comparator B switches from LOW to HIGH and sets the S-R latch, causing the output (pin 3) to go HIGH and turning the discharge transistor Q_1 off. The output will stay HIGH until the normally LOW threshold input goes above $\frac{2}{3}V_{CC}$ and causes the output of comparator A to switch from LOW to HIGH. This resets the latch, causing the output to go back LOW and turning the discharge transistor on. The external reset input can be used to reset the latch independent of the threshold circuit. The trigger and threshold inputs (pins 2 and 6) are controlled by external components connected to produce either monostable or astable action.

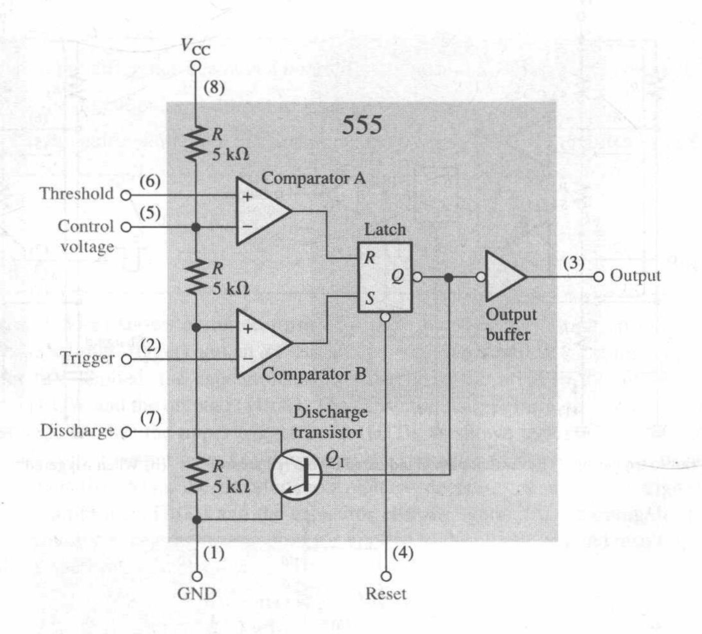


FIGURE 7-52
Internal functional diagram of a 555 timer (pin numbers are in parentheses).

Monostable (One-Shot) Operation

An external resistor and capacitor connected as shown in Figure 7–53 are used to set up the 555 timer as a nonretriggerable one-shot. The pulse width of the output is determined by the time constant of R_1 and C_1 according to the following formula:

$$t_W = 1.1R_1C_1 \qquad \text{Equation 7–3}$$

The control voltage input is not used and is connected to a decoupling capacitor C_2 to prevent noise from affecting the trigger and threshold levels.

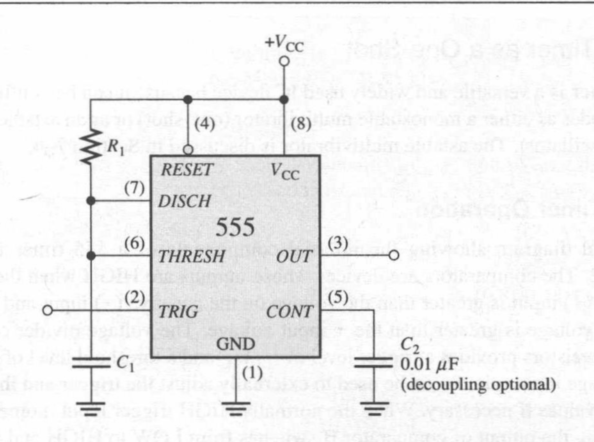
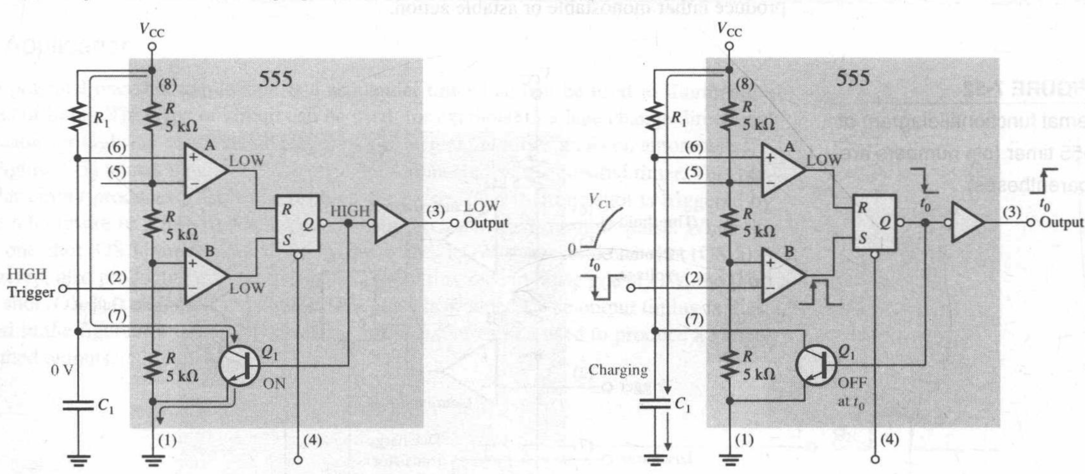
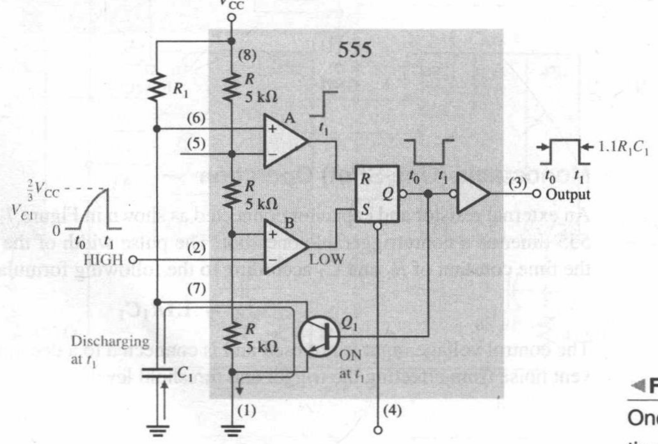


FIGURE 7-53 The 555 timer connected as a one-shot.

Before a trigger pulse is applied, the output is LOW and the discharge transistor Q_1 is *on*, keeping C_1 discharged as shown in Figure 7–54(a). When a negative-going trigger pulse is applied at t_0, the output goes HIGH and the discharge transistor turns *off*, allowing capacitor C_1 to begin charging through R_1 as shown in part (b). When C_1 charges to $^1/_3\ V_{CC}$, the output goes back LOW at t_1 and Q_1 turns *on* immediately, discharging C_1 as shown in part (c). As you can see, the charging rate of C_1 determines how long the output is HIGH.



(a) Prior to triggering. (The current path is indicated by the red arrow.) (b) When triggered



(c) At end of charging interval

FIGURE 7-54 One-shot operation of the 555 timer.

EXAMPLE 7-13

What is the output pulse width for a 555 monostable circuit with $R_1 = 2.2\ \text{k}\Omega$ and $C_1 = 0.01\ \mu\text{F}$?

Solution

From Equation 7-3 the pulse width is

$$t_W = 1.1R_1C_1 = 1.1(2.2\ \text{k}\Omega)(0.01\ \mu\text{F}) = \mathbf{24.2\ \mu s}$$

Related Problem

For $C_1 = 0.01\ \mu\text{F}$, determine the value of R_1 for a pulse width of 1 ms.

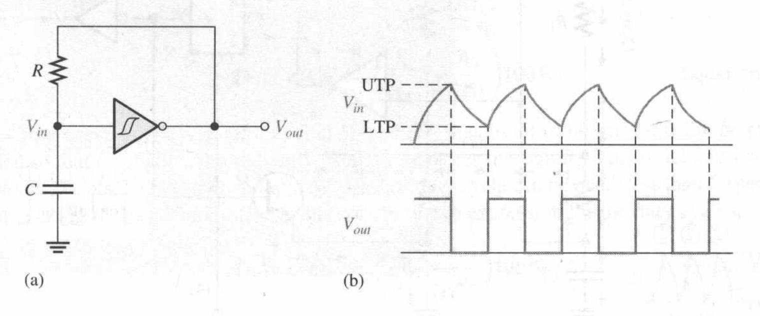
In normal operation, a one-shot produces only a single pulse, which can be difficult to measure on an oscilloscope because the pulse does not occur regularly. To obtain a stable display for test purposes, it is useful to trigger the one-shot from a pulse generator that is set to a longer period than the expected pulse width and trigger the oscilloscope from the same pulse. For very long pulses, either store the waveform using a digital storage oscilloscope or shorten the time constant by some known factor. For example, replace a 1000 μF capacitor with a 1 μF capacitor to shorten the time by a factor of 1000. A faster pulse is easier to see and measure with an oscilloscope.

SECTION 7-5 CHECKUP

1. Describe the difference between a nonretriggerable and a retriggerable one-shot.
2. How is the output pulse width set in most IC one-shots?
3. What is the pulse width of a 555 timer one-shot when $C = 1\ \mu\text{F}$ and $R = 10\ \text{k}\Omega$?

7-6 The Astable Multivibrator

Figure 7-55(a) shows a simple form of astable multivibrator using an inverter with hysteresis (Schmitt trigger) and an *RC* circuit connected in a feedback arrangement. When power is first applied, the capacitor has no charge; so the input to the Schmitt trigger inverter is LOW and the output is HIGH. The capacitor charges through *R* until the inverter input voltage reaches the upper trigger point (UTP), as shown in Figure 7-55(b). At this point, the inverter output goes LOW, causing the capacitor to discharge back through *R*, shown in part (b). When the inverter input voltage decreases to the lower trigger point (LTP), its output goes HIGH and the capacitor charges again. This charging/discharging cycle continues to repeat as long as power is applied to the circuit, and the resulting output is a pulse waveform, as indicated.



▲ **FIGURE 7-55**

Basic astable multivibrator using a Schmitt trigger.

The 555 Timer as an Astable Multivibrator

A 555 timer connected to operate as an astable multivibrator is shown in Figure 7–56. Notice that the threshold input (*THRESH*) is now connected to the trigger input (*TRIG*). The external components R_1, R_2, and C_1 form the timing network that sets the frequency of oscillation. The 0.01 μF capacitor, C_2, connected to the control (*CONT*) input is strictly for decoupling and has no effect on the operation; in some cases it can be left off.

InfoNote

Most systems require a timing source to provide accurate clock waveforms. The timing section controls all system timing and is responsible for the proper operation of the system hardware. The timing section usually consists of a crystal-controlled oscillator and counters for frequency division. Using a high-frequency oscillator divided down to a lower frequency provides for greater accuracy and frequency stability.

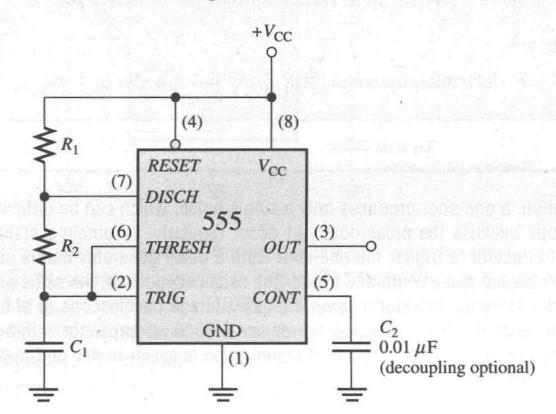


FIGURE 7-56 The 555 timer connected as an astable multivibrator (oscillator).

Initially, when the power is turned on, the capacitor (C_1) is uncharged and thus the trigger voltage (pin 2) is at 0 V. This causes the output of comparator B to be HIGH and the output of comparator A to be LOW, forcing the output of the latch, and thus the base of Q_1, LOW and keeping the transistor off. Now, C_1 begins charging through R_1 and R_2, as indicated in Figure 7–57. When the capacitor voltage reaches $^1/_3\,V_{CC}$, comparator B switches to its LOW output state; and when the capacitor voltage reaches $^2/_3\,V_{CC}$, comparator A switches to its HIGH output state. This resets the latch, causing the base of Q_1 to go HIGH and turning on the transistor. This sequence creates a discharge path for the capacitor through R_2 and the transistor, as indicated. The capacitor now begins to discharge, causing comparator A to go LOW. At the point where the capacitor discharges down to $^1/_3\,V_{CC}$, comparator B switches HIGH; this sets the latch, making the base of Q_1 LOW and turning off the transistor. Another charging cycle begins, and the entire process repeats. The

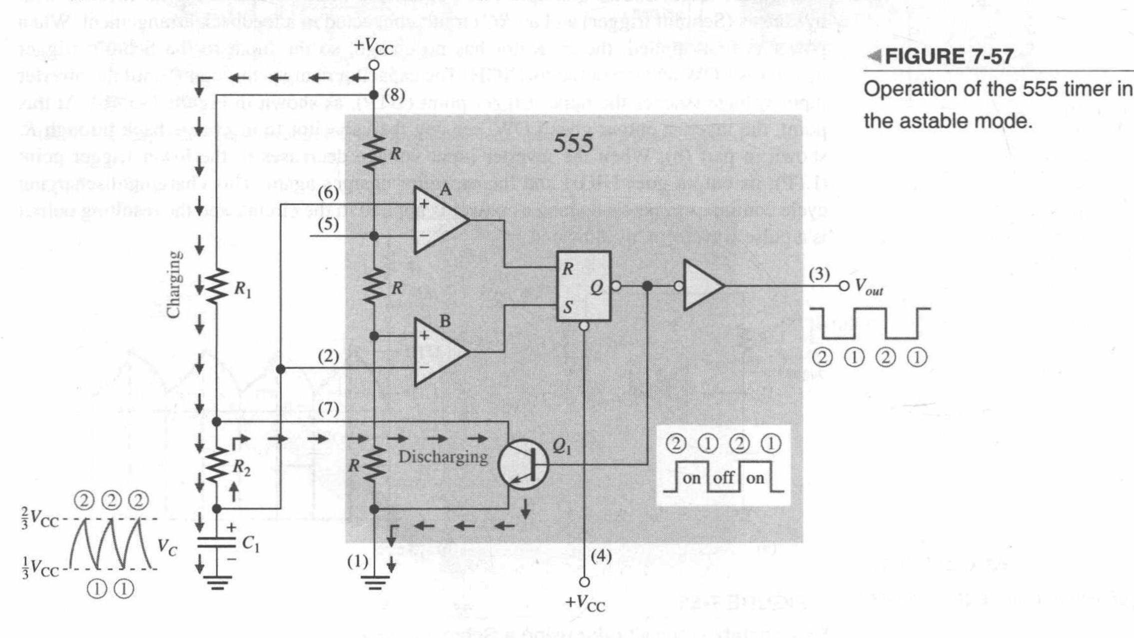
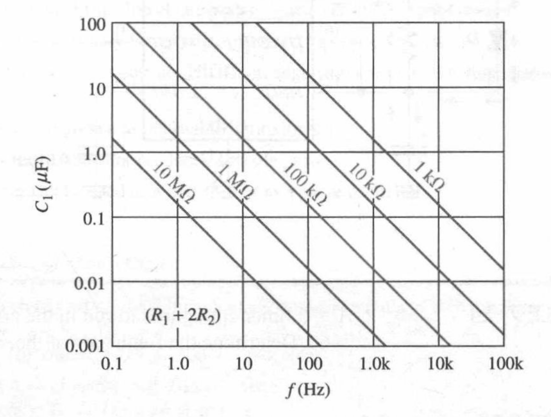


FIGURE 7-57 Operation of the 555 timer in the astable mode.

result is a rectangular wave output whose duty cycle depends on the values of R_1 and R_2. The frequency of oscillation is given by the following formula, or it can be found using the graph in Figure 7–58.

$$f = \frac{1.44}{(R_1 + 2R_2)C_1} \quad \text{Equation 7–4}$$



▲ **FIGURE 7-58**

Frequency of oscillation as a function of C_1 and $R_1 + 2R_2$. The sloped lines are values of $R_1 + 2R_2$.

By selecting R_1 and R_2, the duty cycle of the output can be adjusted. Since C_1 charges through $R_1 + R_2$ and discharges only through R_2, duty cycles approaching a minimum of 50 percent can be achieved if $R_2 \gg R_1$ so that the charging and discharging times are approximately equal.

An expression for the duty cycle is developed as follows. The time that the output is HIGH (t_H) is how long it takes C_1 to charge from $\frac{1}{3} V_{CC}$ to $\frac{2}{3} V_{CC}$. It is expressed as

$$t_H = 0.7(R_1 + R_2)C_1 \quad \text{Equation 7–5}$$

The time that the output is LOW (t_L) is how long it takes C_1 to discharge from $\frac{1}{3} V_{CC}$ to $\frac{2}{3} V_{CC}$. It is expressed as

$$t_L = 0.7R_2C_1 \quad \text{Equation 7–6}$$

The period, T, of the output waveform is the sum of t_H and t_L. This is the reciprocal of f in Equation 7–4.

$$T = t_H + t_L = 0.7(R_1 + 2R_2)C_1$$

Finally, the duty cycle is

$$\text{Duty cycle} = \frac{t_H}{T} = \frac{t_H}{t_H + t_L}$$

$$\textbf{Duty cycle} = \left(\frac{R_1 + R_2}{R_1 + 2R_2}\right)100\% \quad \text{Equation 7–7}$$

To achieve duty cycles of less than 50 percent, the circuit in Figure 7–56 can be modified so that C_1 charges through only R_1 and discharges through R_2. This is achieved with a diode, D_1, placed as shown in Figure 7–59. The duty cycle can be made less than 50 percent by making R_1 less than R_2. Under this condition, the expression for the duty cycle is

$$\textbf{Duty cycle} = \left(\frac{R_1}{R_1 + R_2}\right)100\% \quad \text{Equation 7–8}$$

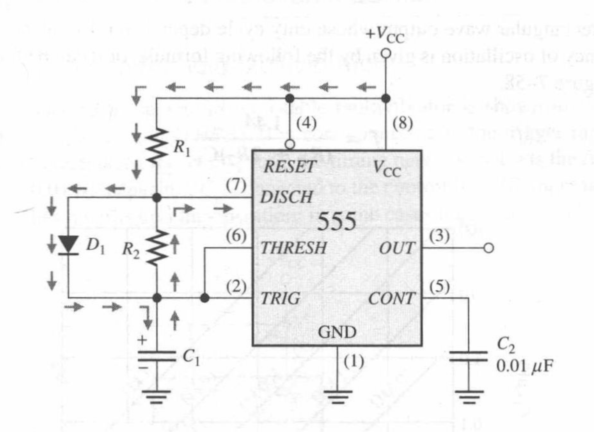


FIGURE 7-59

The addition of diode D_1 allows the duty cycle of the output to be adjusted to less than 50 percent by making $R_1 < R_2$.

EXAMPLE 7-14

A 555 timer configured to run in the astable mode (pulse oscillator) is shown in Figure 7–60. Determine the frequency of the output and the duty cycle.

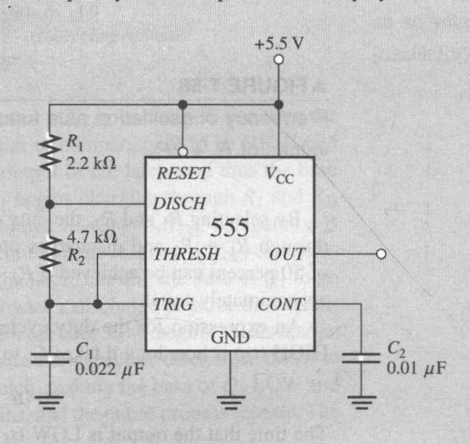


FIGURE 7-60

Open file F07-60 to verify operation.

MultiSim

Solution

Use Equations 7–4 and 7–7.

$$f = \frac{1.44}{(R_1 + 2R_2)C_1} = \frac{1.44}{(2.2\text{ k}\Omega + 9.4\text{ k}\Omega)0.022\ \mu\text{F}} = \mathbf{5.64\ kHz}$$

$$\text{Duty cycle} = \left(\frac{R_1 + R_2}{R_1 + 2R_2}\right)100\% = \left(\frac{2.2\text{ k}\Omega + 4.7\text{ k}\Omega}{2.2\text{ k}\Omega + 9.4\text{ k}\Omega}\right)100\% = \mathbf{59.5\%}$$

Related Problem

Determine the duty cycle in Figure 7–60 if a diode is connected across R_2 as indicated in Figure 7–59.

SECTION 7-6 CHECKUP

1. Explain the difference in operation between an astable multivibrator and a monostable multivibrator.
2. For a certain astable multivibrator, $t_H = 15$ ms and $T = 20$ ms. What is the duty cycle of the output?

TRUE/FALSE QUIZ

Answers are at the end of the chapter.

1. A latch has two stable states.
2. A latch is considered to be in the SET state when the Q output is LOW.
3. A gated D latch must be enabled in order to change state.
4. Flip-flops and latches are both bistable devices.
5. An edge-triggered D flip-flop changes state whenever the D input changes.
6. A clock input is necessary for an edge-triggered flip-flop.
7. When both the J and K inputs are HIGH, an edge-triggered J-K flip-flop changes state on each clock pulse.
8. A one-shot is also known as an astable multivibrator.
9. When triggered, a one-shot produces a single pulse.
10. The 555 timer can be used as a one-shot or as a pulse oscillator.

SELF-TEST

Answers are at the end of the chapter.

1. If an S-R latch has a 1 on the S input and a 0 on the R input and then the S input goes to 0, the latch will be
   (a) set **(b)** reset **(c)** invalid **(d)** clear
2. The invalid state of an S-R latch occurs when
   (a) $S = 1, R = 0$ **(b)** $S = 0, R = 1$
   (c) $S = 1, R = 1$ **(d)** $S = 0, R = 0$
3. For a gated D latch, the Q output always equals the D input
   (a) before the enable pulse
   (b) during the enable pulse
   (c) immediately after the enable pulse
   (d) answers (b) and (c)
4. Like the latch, the flip-flop belongs to a category of logic circuits known as
   (a) monostable multivibrators
   (b) bistable multivibrators
   (c) astable multivibrators
   (d) one-shots
5. The purpose of the clock input to a flip-flop is to
   (a) clear the device
   (b) set the device
   (c) always cause the output to change states
   (d) cause the output to assume a state dependent on the controlling (J-K or D) inputs.
6. For an edge-triggered D flip-flop,
   (a) a change in the state of the flip-flop can occur only at a clock pulse edge
   (b) the state that the flip-flop goes to depends on the D input
   (c) the output follows the input at each clock pulse
   (d) all of these answers
7. A feature that distinguishes the J-K flip-flop from the D flip-flop is the
   (a) toggle condition **(b)** preset input
   (c) type of clock **(d)** clear input
8. A flip-flop is in the toggle condition when
   (a) $J = 1, K = 0$ **(b)** $J = 1, K = 1$
   (c) $J = 0, K = 0$ **(d)** $J = 0, K = 1$
9. A J-K flip-flop with $J = 1$ and $K = 1$ has a 10 kHz clock input. The Q output is
   (a) constantly HIGH **(b)** constantly LOW
   (c) a 10 kHz square wave **(d)** a 5 kHz square wave
10. A one-shot is a type of
    (a) monostable multivibrator **(b)** astable multivibrator
    (c) timer **(d)** answers (a) and (c)
    (e) answers (b) and (c)
11. The output pulse width of a nonretriggerable one-shot depends on
    (a) the trigger intervals **(b)** the supply voltage
    (c) a resistor and capacitor **(d)** the threshold voltage

12. An astable multivibrator
    (a) requires a periodic trigger input
    (b) has no stable state
    (c) is an oscillator
    (d) produces a periodic pulse output
    (e) answers (a), (b), (c), and (d)
    (f) answers (b), (c), and (d) only

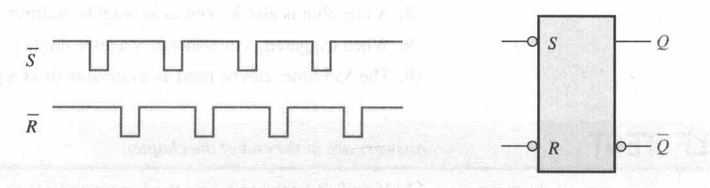
PROBLEMS

Answers to odd-numbered problems are at the end of the book.

Section 7–1 Latches

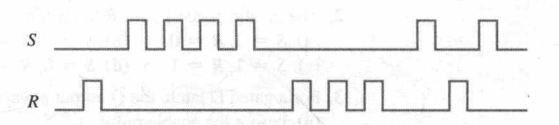
1. If the waveforms in Figure 7–61 are applied to an active-LOW input $\overline{S}$-$\overline{R}$ latch, draw the resulting Q output waveform in relation to the inputs. Assume that Q starts LOW.

▶ FIGURE 7-61



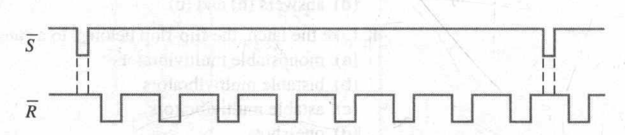
2. Solve Problem 1 for the input waveforms in Figure 7–62 applied to an active-HIGH S-R latch.

▶ FIGURE 7-62



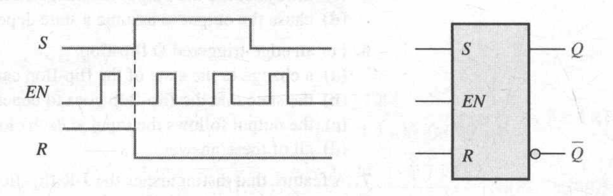
3. Solve Problem 1 for the input waveforms in Figure 7–63.

▶ FIGURE 7-63



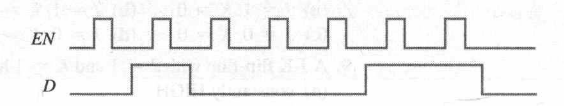
4. For a gated S-R latch, determine the Q and $\overline{Q}$ outputs for the inputs in Figure 7–64. Show them in proper relation to the enable input. Assume that Q starts LOW.

▶ FIGURE 7-64



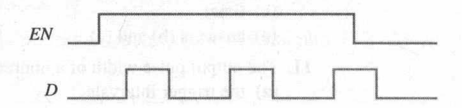
5. Determine the output of a gated D latch for the inputs in Figure 7–65.

▶ FIGURE 7-65



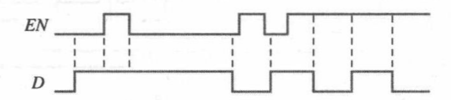
6. Determine the output of a gated D latch for the inputs in Figure 7–66.

▶ FIGURE 7-66



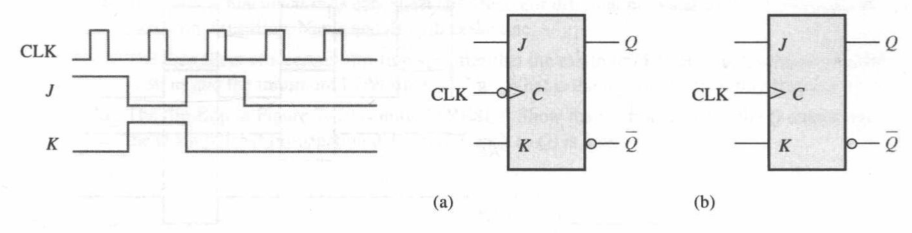
7. For a gated D latch, the waveforms shown in Figure 7–67 are observed on its inputs. Draw the timing diagram showing the output waveform you would expect to see at Q if the latch is initially RESET.

▶ FIGURE 7-67



Section 7–2 Flip-Flops

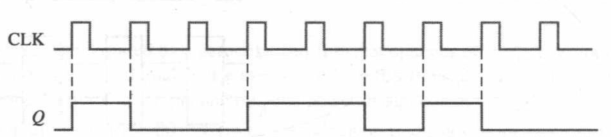
8. Two edge-triggered J-K flip-flops are shown in Figure 7–68. If the inputs are as shown, draw the Q output of each flip-flop relative to the clock, and explain the difference between the two. The flip-flops are initially RESET.



▲ FIGURE 7-68

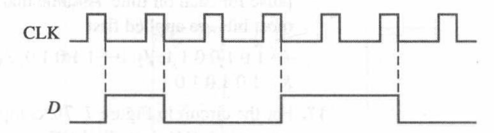
9. The Q output of an edge-triggered D flip-flop is shown in relation to the clock signal in Figure 7–69. Determine the input waveform on the D input that is required to produce this output if the flip-flop is a positive edge-triggered type.

▶ FIGURE 7-69



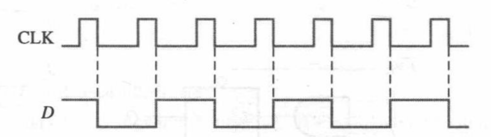
10. Draw the Q output relative to the clock for a D flip-flop with the inputs as shown in Figure 7–70. Assume positive edge-triggering and Q initially LOW.

▶ FIGURE 7-70



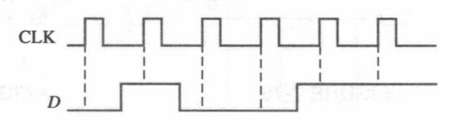
11. Solve Problem 10 for the inputs in Figure 7–71.

▶ FIGURE 7-71



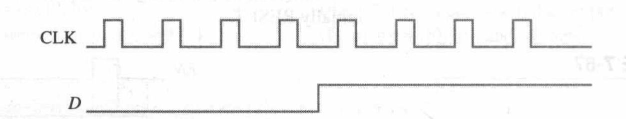
12. For a positive edge-triggered D flip-flop with the input as shown in Figure 7–72, determine the Q output relative to the clock. Assume that Q starts LOW.

▶ FIGURE 7-72



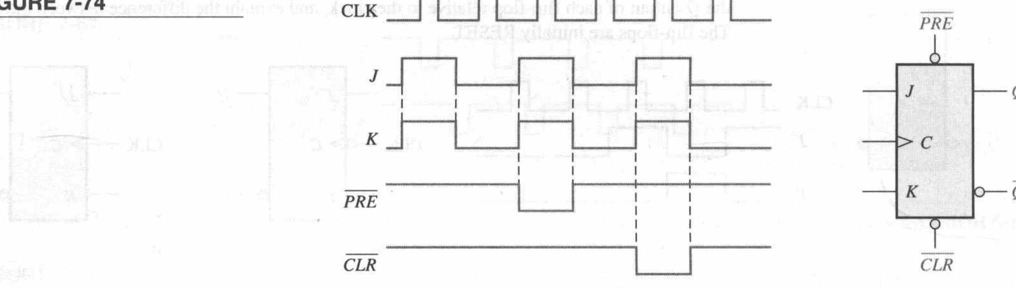
13. Solve Problem 12 for the input in Figure 7–73.

FIGURE 7-73



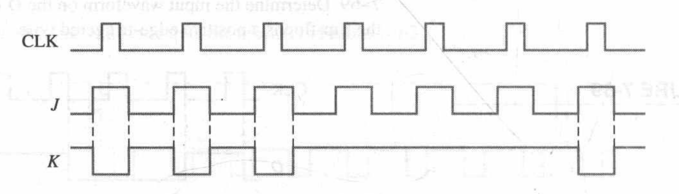
14. Determine the Q waveform relative to the clock if the signals shown in Figure 7–74 are applied to the inputs of the J-K flip-flop. Assume that Q is initially LOW.

FIGURE 7-74



15. For a negative edge-triggered J-K flip-flop with the inputs in Figure 7–75, develop the Q output waveform relative to the clock. Assume that Q is initially LOW.

FIGURE 7-75



16. The following serial data are applied to the flip-flop through the AND gates as indicated in Figure 7–76. Determine the resulting serial data that appear on the Q output. There is one clock pulse for each bit time. Assume that Q is initially 0 and that $\overline{PRE}$ and $\overline{CLR}$ are HIGH. Rightmost bits are applied first.

J_1: 1 0 1 0 0 1 1; J_2: 0 1 1 1 0 1 0; J_3: 1 1 1 1 0 0 0; K_1: 0 0 0 1 1 1 0; K_2: 1 1 0 1 1 0 0; K_3: 1 0 1 0 1 0 1

17. For the circuit in Figure 7–76, complete the timing diagram in Figure 7–77 by showing the Q output (which is initially LOW). Assume $\overline{PRE}$ and $\overline{CLR}$ remain HIGH.

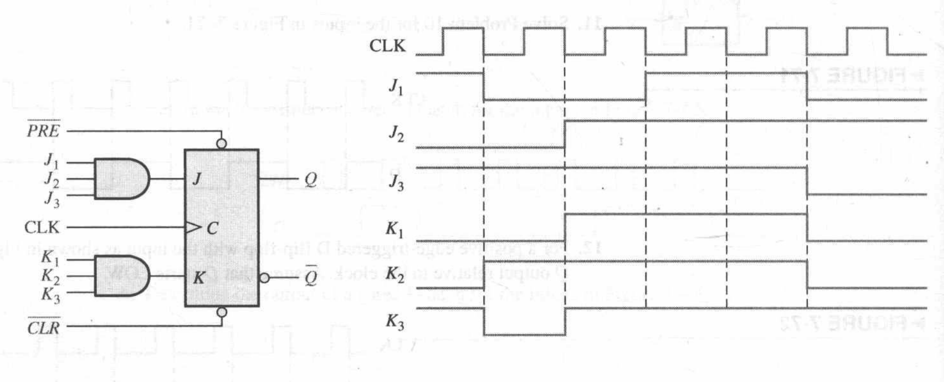
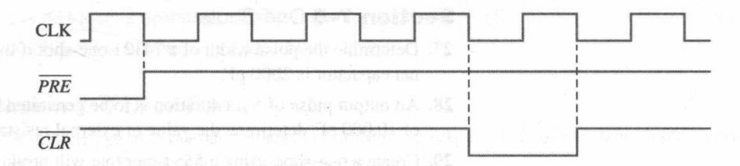


FIGURE 7-76

FIGURE 7-77

18. Solve Problem 17 with the same J and K inputs but with the $\overline{PRE}$ and $\overline{CLR}$ inputs as shown in Figure 7–78 in relation to the clock.

► **FIGURE 7-78**



Section 7–3 Flip-Flop Operating Characteristics

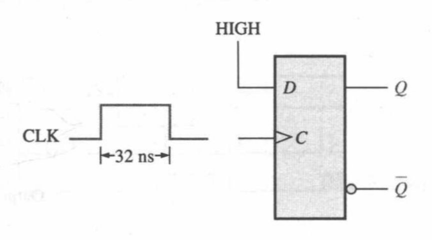
19. What determines the power dissipation of a flip-flop?

20. Typically, a manufacturer's data sheet specifies four different propagation delay times associated with a flip-flop. Name and describe each one.

21. The data sheet of a certain flip-flop specifies that the minimum HIGH time for the clock pulse is 30 ns and the minimum LOW time is 37 ns. What is the maximum operating frequency?

22. The flip-flop in Figure 7–79 is initially RESET. Show the relation between the Q output and the clock pulse if propagation delay t_{PLH} (clock to Q) is 8 ns.

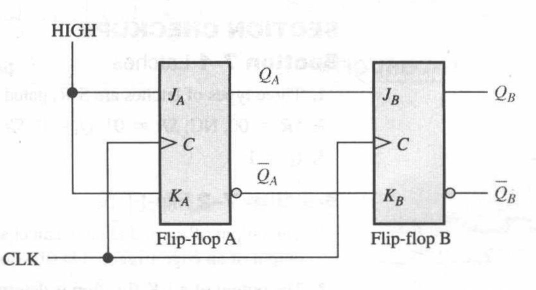
► **FIGURE 7-79**



23. The direct current required by a particular flip-flop that operates on a +5 V dc source is found to be 10 mA. A certain digital device uses 15 of these flip-flops. Determine the current capacity required for the +5 V dc supply and the total power dissipation of the system.

24. For the circuit in Figure 7–80, determine the maximum frequency of the clock signal for reliable operation if the set-up time for each flip-flop is 2 ns and the propagation delays (t_{PLH} and t_{PHL}) from clock to output are 5 ns for each flip-flop.

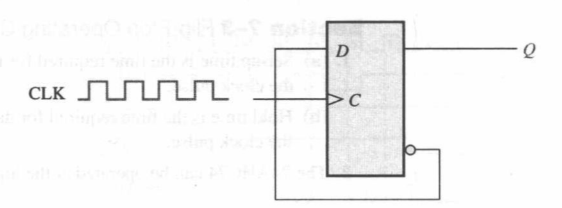
► **FIGURE 7-80**



Section 7–4 Flip-Flop Applications

25. A D flip-flop is connected as shown in Figure 7–81. Determine the Q output in relation to the clock. What specific function does this device perform?

► **FIGURE 7-81**



26. For the circuit in Figure 7–80, develop a timing diagram for eight clock pulses, showing the Q_A and Q_B outputs in relation to the clock.

Section 7–5 One-Shots

27. Determine the pulse width of a 74121 one-shot if the external resistor is 3.3 kΩ and the external capacitor is 2000 pF.

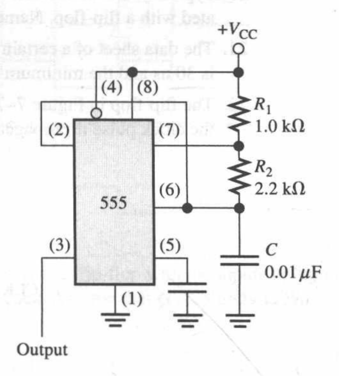
28. An output pulse of 5 μs duration is to be generated by a 74LS122 one-shot. Using a capacitor of 10,000 pF, determine the value of external resistance required.

29. Create a one-shot, using a 555 timer that will produce a 0.25 s output pulse.

Section 7–6 The Astable Multivibrator

30. A 555 timer is configured to run as an astable multivibrator as shown in Figure 7–82. Determine its frequency.

▶ FIGURE 7-82



31. Determine the values of the external resistors for a 555 timer used as an astable multivibrator with an output frequency of 20 kHz, if the external capacitor C is 0.002 μF and the duty cycle is to be approximately 75%.

ANSWERS

SECTION CHECKUPS

Section 7–1 Latches

1. Three types of latches are S-R, gated S-R, and gated D.
2. $SR = 00$, NC; $SR = 01$, $Q = 0$; $SR = 10$, $Q = 1$; $SR = 11$, invalid
3. $Q = 1$

Section 7–2 Flip-Flops

1. The output of a gated D latch can change any time the gate enable (*EN*) input is active. The output of an edge-triggered D flip-flop can change only on the triggering edge of a clock pulse.
2. The output of a J-K flip-flop is determined by the state of its two inputs whereas the output of a D flip-flop follows the input.
3. Output Q goes HIGH on the trailing edge of the first clock pulse, LOW on the trailing edge of the second pulse, HIGH on the trailing edge of the third pulse, and LOW on the trailing edge of the fourth pulse.

Section 7–3 Flip-Flop Operating Characteristics

1. **(a)** Set-up time is the time required for input data to be present before the triggering edge of the clock pulse.

   (b) Hold time is the time required for data to remain on the inputs after the triggering edge of the clock pulse.
2. The 74AHC74 can be operated at the highest frequency, according to Table 7–4.

Section 7–4 Flip-Flop Applications

1. A group of data storage flip-flops is a register.
2. For divide-by-2 operation, the flip-flop must toggle ($D = \overline{Q}$).
3. Six flip-flops are used in a divide-by-64 device.

Section 7–5 One-Shots

1. A nonretriggerable one-shot times out before it can respond to another trigger input. A retriggerable one-shot responds to each trigger input.
2. Pulse width is set with external R and C components.
3. 11 ms.

Section 7–6 The Astable Multivibrator

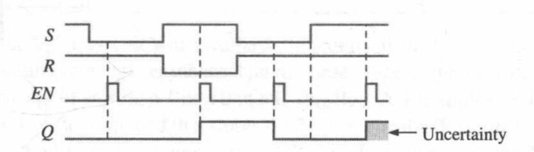
1. An astable multivibrator has no stable state. A monostable multivibrator has one stable state.
2. Duty cycle = (15 ms/20 ms)100% = 75%

RELATED PROBLEMS FOR EXAMPLES

7–1 The Q output is the same as shown in Figure 7–5(b).

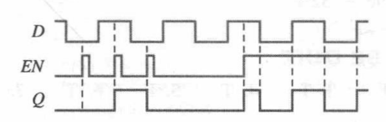
7–2 See Figure 7–83.

► FIGURE 7-83

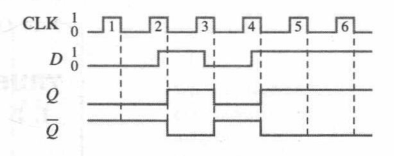


7–3 See Figure 7–84.

7–4 See Figure 7–85.



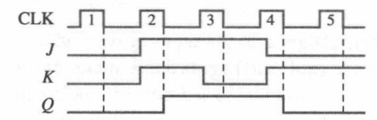
▲ FIGURE 7-84



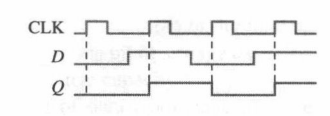
▲ FIGURE 7-85

7–5 See Figure 7–86.

7–6 See Figure 7–87.



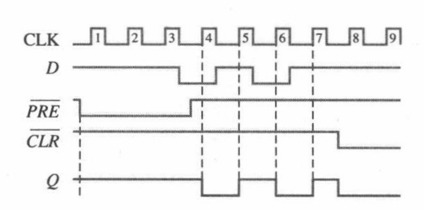
▲ FIGURE 7-86



▲ FIGURE 7-87

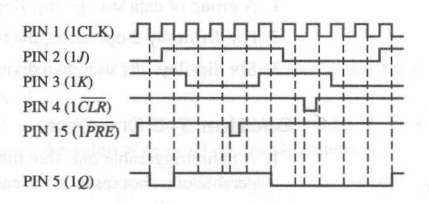
7–7 See Figure 7–88.

► FIGURE 7-88



7–8 See Figure 7–89.

▶ **FIGURE 7-89**



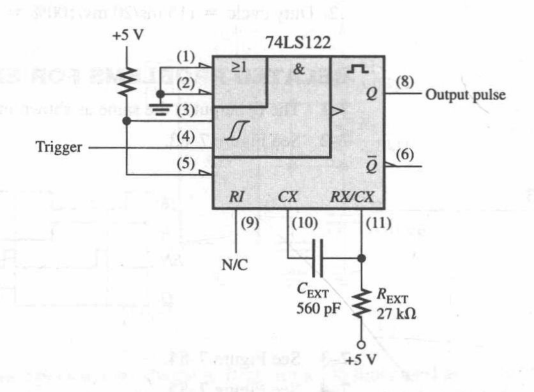
7–9 $2^5 = 32$. Five flip-flops are required.

7–10 Sixteen states require four flip-flops ($2^4 = 16$).

7–11 $C_{EXT} = 7143$ pF connected from *CX* to *RX/CX* of the 74121 with no external resistor.

7–12 $C_{EXT} = 560$ pF, $R_{EXT} = 27$ kΩ. See Figure 7–90.

▶ **FIGURE 7-90**



7–13 $R_1 = 91$ kΩ

7–14 Duty cycle ≅ 32%

TRUE/FALSE QUIZ

1. T **2.** F **3.** T **4.** T **5.** F **6.** T **7.** T **8.** F **9.** T **10.** T

SELF-TEST

1. (a) **2.** (c) **3.** (d) **4.** (b) **5.** (d) **6.** (d)

7. (a) **8.** (b) **9.** (d) **10.** (d) **11.** (c) **12.** (f)

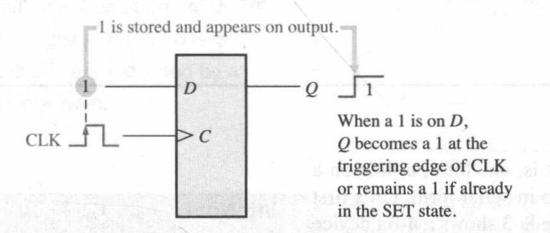
Chapter 8 Shift Registers

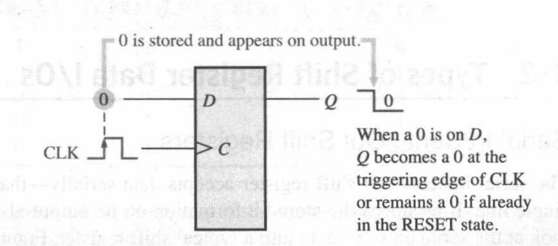
CHAPTER OUTLINE

8-1 Shift Register Operations

A register can consist of one or more flip-flops used to store and shift data.

A **register** is a digital circuit with two basic functions: data storage and data movement. The storage capability of a register makes it an important type of memory device. Figure 8–1 illustrates the concept of storing a 1 or a 0 in a D flip-flop. A 1 is applied to the data input as shown, and a clock pulse is applied that stores the 1 by *setting* the flip-flop. When the 1 on the input is removed, the flip-flop remains in the SET state, thereby storing the 1. A similar procedure applies to the storage of a 0 by *resetting* the flip-flop, as also illustrated in Figure 8–1.



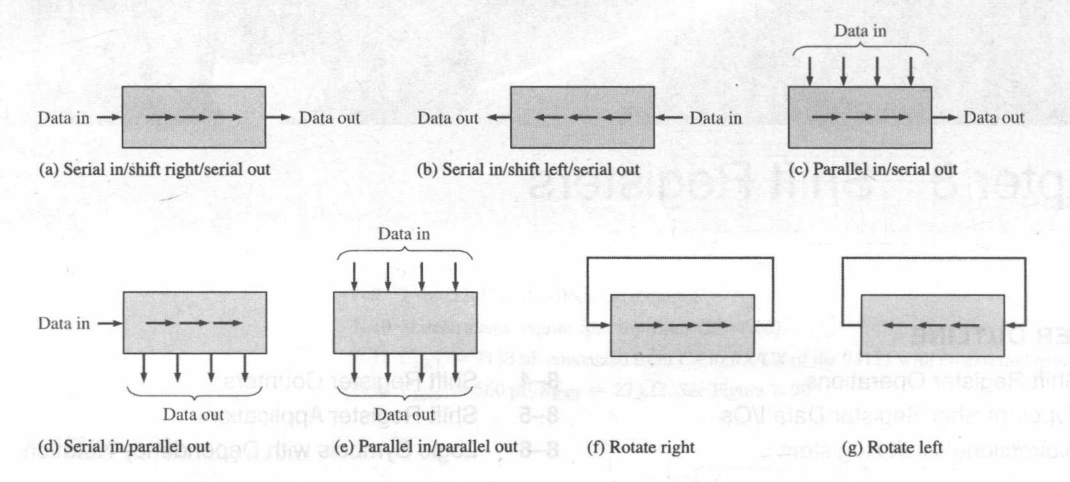


▲ **FIGURE 8-1**

The flip-flop as a storage element.

The *storage capacity* of a register is the total number of bits (1s and 0s) of digital data it can retain. Each **stage** (flip-flop) in a shift register represents one bit of storage capacity; therefore, the number of stages in a register determines its storage capacity.

The *shift capability* of a register permits the movement of data from stage to stage within the register or into or out of the register upon application of clock pulses. Figure 8–2 illustrates the types of data movement in shift registers. The block represents any arbitrary 4-bit register, and the arrows indicate the direction of data movement.



▲ **FIGURE 8-2**

Basic data movement in shift registers. (Four bits are used for illustration. The bits move in the direction of the arrows.)

SECTION 8-1 CHECKUP

Answers are at the end of the chapter.

1. What determines the storage capacity of a shift register?
2. What two principal functions are performed by a shift register?

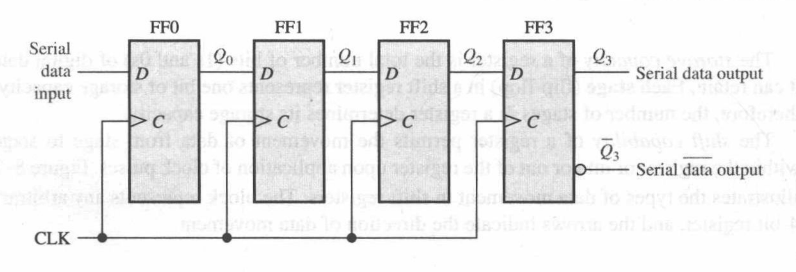
8-2 Types of Shift Register Data I/Os

Serial In/Serial Out Shift Registers

The serial in/serial out shift register accepts data serially—that is, one bit at a time on a single line. It produces the stored information on its output also in serial form. Let's first look at the serial entry of data into a typical shift register. Figure 8–3 shows a 4-bit device implemented with D flip-flops. With four stages, this register can store up to four bits of data.

InfoNote

Frequently, it is necessary to *clear* an internal register in a processor. For example, a register may be cleared prior to an arithmetic or other operation. One way that registers in a processor are cleared is using software to subtract the contents of the register from itself. The result, of course, will always be zero. For example, a processor instruction that performs this operation is SUB AL,AL. with this instruction, the register named AL is cleared.



▲ **FIGURE 8-3**

Serial in/serial out shift register.

Table 8–1 shows the entry of the four bits 1010 into the register in Figure 8–3, beginning with the least significant bit. The register is initially clear. The 0 is put onto the data input line, making $D = 0$ for FF0. When the first clock pulse is applied, FF0 is reset, thus storing the 0.

TABLE 8-1

Shifting a 4-bit code into the shift register in Figure 8–3. Data bits are indicated by a beige screen.

CLK	FF0 (Q_0)	FF1 (Q_1)	FF2 (Q_2)	FF3 (Q_3)
Initial	0	0	0	0
1	**0**	0	0	0
2	**1**	**0**	0	0
3	**0**	**1**	**0**	0
4	**1**	**0**	**1**	**0**

Next the second bit, which is a 1, is applied to the data input, making $D = 1$ for FF0 and $D = 0$ for FF1 because the D input of FF1 is connected to the Q_0 output. When the second clock pulse occurs, the 1 on the data input is shifted into FF0, causing FF0 to set; and the 0 that was in FF0 is shifted into FF1.

The third bit, a 0, is now put onto the data-input line, and a clock pulse is applied. The 0 is entered into FF0, the 1 stored in FF0 is shifted into FF1, and the 0 stored in FF1 is shifted into FF2.

For serial data, one bit at a time is transferred.

The last bit, a 1, is now applied to the data input, and a clock pulse is applied. This time the 1 is entered into FF0, the 0 stored in FF0 is shifted into FF1, the 1 stored in FF1 is shifted into FF2, and the 0 stored in FF2 is shifted into FF3. This completes the serial entry of the four bits into the shift register, where they can be stored for any length of time as long as the flip-flops have dc power.

If you want to get the data out of the register, the bits must be shifted out serially to the Q_3 output, as Table 8–2 illustrates. After CLK4 in the data-entry operation just described, the LSB, 0, appears on the Q_3 output. When clock pulse CLK5 is applied, the second bit appears on the Q_3 output. Clock pulse CLK6 shifts the third bit to the output, and CLK7 shifts the fourth bit to the output. While the original four bits are being shifted out, more bits can be shifted in. All zeros are shown being shifted in, after CLK8.

TABLE 8-2

Shifting a 4-bit code out of the shift register in Figure 8–3. Data bits are indicated by a beige screen.

CLK	FF0 (Q_0)	FF1 (Q_1)	FF2 (Q_2)	FF3 (Q_3)
Initial	**1**	**0**	**1**	**0**
5	0	**1**	**0**	**1**
6	0	0	**1**	**0**
7	0	0	0	**1**
8	0	0	0	0

EXAMPLE 8-1

Show the states of the 5-bit register in Figure 8–4(a) for the specified data input and clock waveforms. Assume that the register is initially cleared (all 0s).

Solution

The first data bit (1) is entered into the register on the first clock pulse and then shifted from left to right as the remaining bits are entered and shifted. The register contains $Q_4Q_3Q_2Q_1Q_0 = 11010$ after five clock pulses. See Figure 8–4(b).

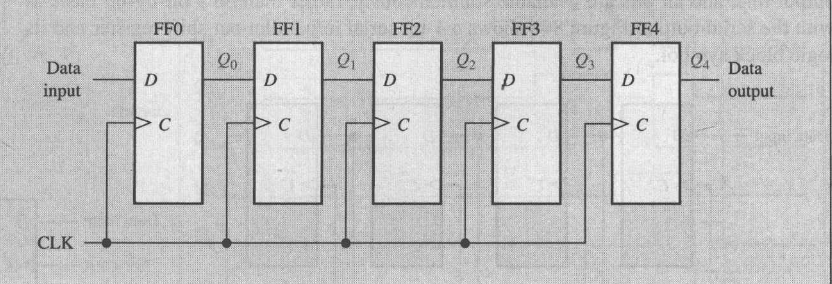


FIGURE 8-4

Open file F08-04 to verify operation. *A multisim tutorial is available on the website.*

MultiSim

Related Problem*

Show the states of the register if the data input is inverted. The register is initially cleared.

*Answers are at the end of the chapter.

A traditional logic block symbol for an 8-bit serial in/serial out shift register is shown in Figure 8–5. The "SRG 8" designation indicates a shift register (SRG) with an 8-bit capacity.

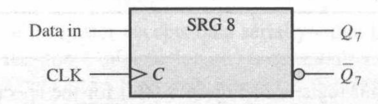
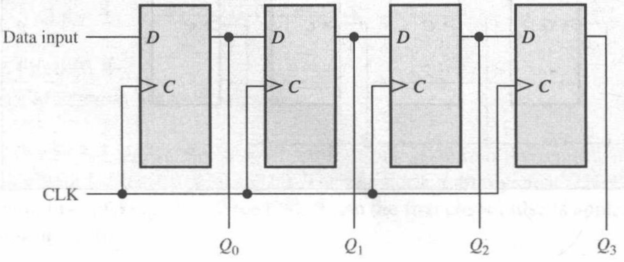


FIGURE 8-5

Logic symbol for an 8-bit serial in/serial out shift register.

Serial In/Parallel Out Shift Registers

Data bits are entered serially (least-significant bit first) into a serial in/parallel out shift register in the same manner as in serial in/serial out registers. The difference is the way in which the data bits are taken out of the register; in the parallel output register, the output of each stage is available. Once the data are stored, each bit appears on its respective output line, and all bits are available simultaneously, rather than on a bit-by-bit basis as with the serial output. Figure 8–6 shows a 4-bit serial in/parallel out shift register and its logic block symbol.



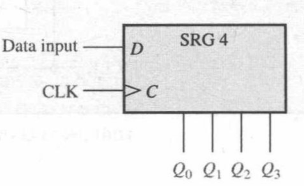
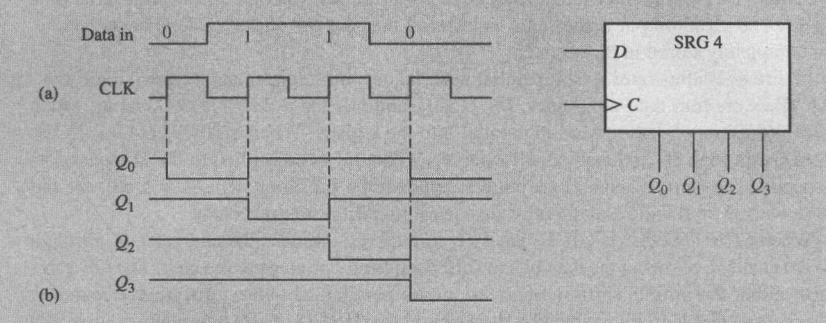


FIGURE 8-6

A serial in/parallel out shift register.

EXAMPLE 8-2

Show the states of the 4-bit register (SRG 4) for the data input and clock waveforms in Figure 8–7(a). The register initially contains all 1s.



▶ FIGURE 8-7

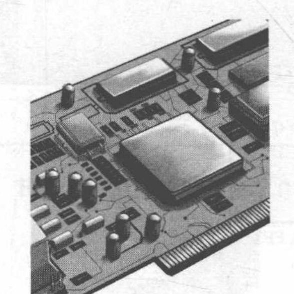
Solution

The register contains 0110 after four clock pulses. See Figure 8–7(b).

Related Problem

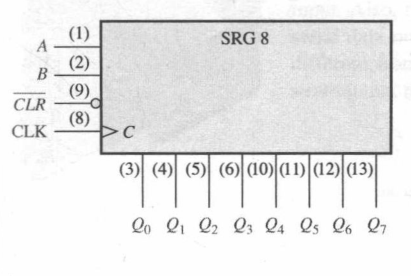
If the data input remains 0 after the fourth clock pulse, what is the state of the register after three additional clock pulses?

IMPLEMENTATION: 8-BIT SERIAL IN/PARALLEL OUT SHIFT REGISTER



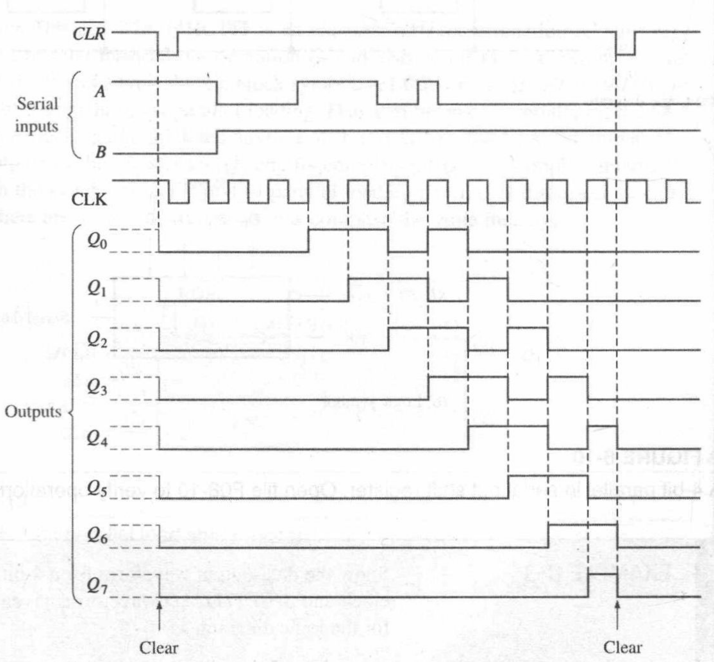
Fixed-Function Device The 74HC164 is an example of a fixed-function IC shift register having serial in/parallel out operation. The logic block symbol is shown in Figure 8–8. This device has two gated serial inputs, A and B, and an asynchronous clear ($\overline{CLR}$) input that is active-LOW. The parallel outputs are Q_0 through Q_7.

A sample timing diagram for the 74HC164 is shown in Figure 8–9. Notice that the serial input data on input A are shifted into and through the register after input B goes HIGH.



▲ FIGURE 8-8

The 74HC164 8-bit serial in/parallel out shift register.



▶ FIGURE 8-9

Sample timing diagram for a 74HC164 shift register.

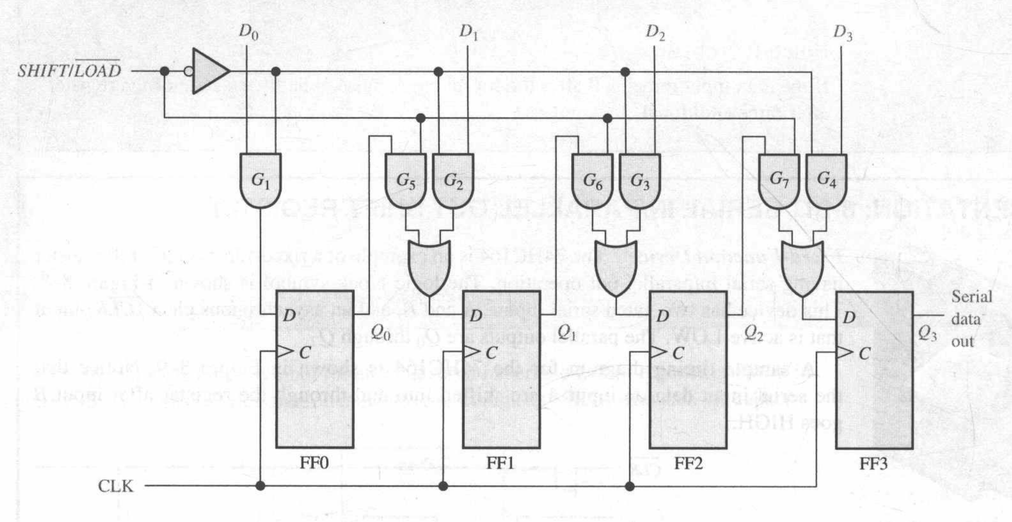
Parallel In/Serial Out Shift Registers

For a register with parallel data inputs, the bits are entered simultaneously into their respective stages on parallel lines rather than on a bit-by-bit basis on one line as with serial data inputs. The serial output is the same as in serial in/serial out shift registers, once the data are completely stored in the register.

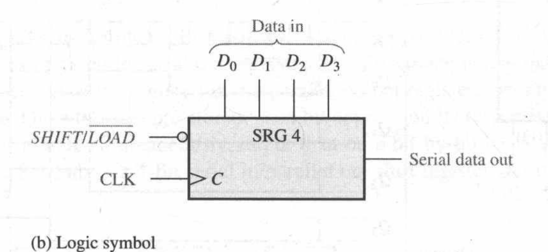
For parallel data, multiple bits are transferred at one time.

Figure 8–10 illustrates a 4-bit parallel in/serial out shift register and a typical logic symbol. There are four data-input lines, D_0, D_1, D_2, and D_3, and a $SHIFT/\overline{LOAD}$ input, which allows four bits of data to **load** in parallel into the register. When $SHIFT/\overline{LOAD}$ is LOW, gates G_1 through G_4 are enabled, allowing each data bit to be applied to the D input of its respective flip-flop. When a clock pulse is applied, the flip-flops with $D = 1$ will set and those with $D = 0$ will reset, thereby storing all four bits simultaneously.

When $SHIFT/\overline{LOAD}$ is HIGH, gates G_1 through G_4 are disabled and gates G_5 through G_7 are enabled, allowing the data bits to shift right from one stage to the next. The OR gates allow either the normal shifting operation or the parallel data-entry operation, depending on which AND gates are enabled by the level on the $SHIFT/\overline{LOAD}$ input. Notice that FF0 has a single AND to disable the parallel input, D_0. It does not require an AND/OR arrangement because there is no serial data in.



(a) Logic diagram



(b) Logic symbol

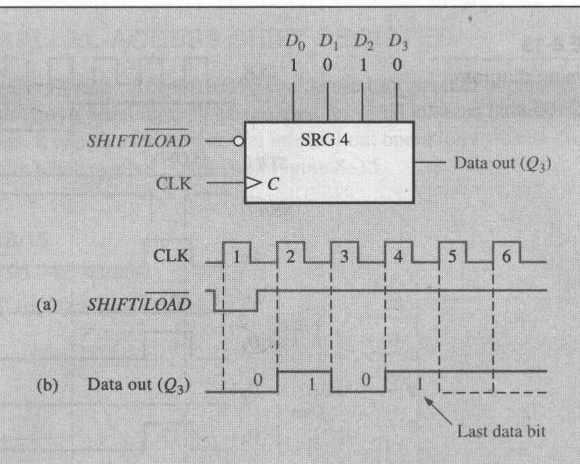
▲ **FIGURE 8-10**

A 4-bit parallel in/serial out shift register. Open file F08-10 to verify operation.

EXAMPLE 8-3

Show the data-output waveform for a 4-bit register with the parallel input data and the clock and $SHIFT/\overline{LOAD}$ waveforms given in Figure 8–11(a). Refer to Figure 8–10(a) for the logic diagram.

FIGURE 8-11



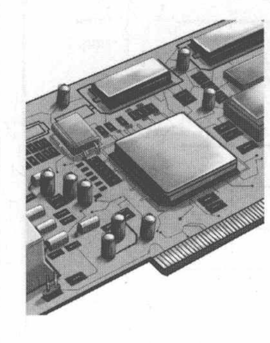
Solution

On clock pulse 1, the parallel data ($D_0D_1D_2D_3$ = 1010) are loaded into the register, making Q_3 a 0. On clock pulse 2 the 1 from Q_2 is shifted onto Q_3; on clock pulse 3 the 0 is shifted onto Q_3; on clock pulse 4 the last data bit (1) is shifted onto Q_3; and on clock pulse 5, all data bits have been shifted out, and only 1s remain in the register (assuming the D_0 input remains a 1). See Figure 8–11(b).

Related Problem

Show the data-output waveform for the clock and *SHIFT/$\overline{LOAD}$* inputs shown in Figure 8–11(a) if the parallel data are $D_0D_1D_2D_3$ = 0101.

IMPLEMENTATION: 8-BIT PARALLEL LOAD SHIFT REGISTER



Fixed-Function Device The 74HC165 is an example of a fixed-function IC shift register that has a parallel in/serial out operation (it can also be operated as serial in/serial out). Figure 8–12 shows a typical logic block symbol. A LOW on the *SHIFT/$\overline{LOAD}$* input (*SH/$\overline{LD}$*) enables asynchronous parallel loading. Data can be entered serially on the *SER* input. Also, the clock can be inhibited anytime with a HIGH on the *CLK INH* input. The serial data outputs of the register are Q_7 and its complement $\overline{Q}_7$. This implementation is different from the synchronous method of parallel loading previously discussed, demonstrating that there are usually several ways to accomplish the same function.

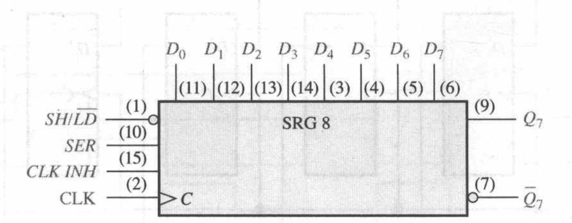


FIGURE 8-12
The 74HC165 8-bit parallel load shift register.

Figure 8–13 is a timing diagram showing an example of the operation of a 74HC165 shift register.

FIGURE 8-13

Sample timing diagram for a 74HC165 shift register.

CLK
CLK INH
SER 0 (LOW)
$SH/\overline{LD}$
Data: D_0 1, D_1 0, D_2 1, D_3 0, D_4 1, D_5 0, D_6 1, D_7 1
Q_7: 1 1 0 1 0 1 0 1
$\overline{Q}_7$: 0 0 1 0 1 0 1 0
Load | ← Inhibit → | ← Serial shift →

Parallel In/Parallel Out Shift Registers

Parallel entry and parallel output of data have been discussed. The parallel in/parallel out register employs both methods. Immediately following the simultaneous entry of all data bits, the bits appear on the parallel outputs. Figure 8–14 shows a parallel in/parallel out shift register.

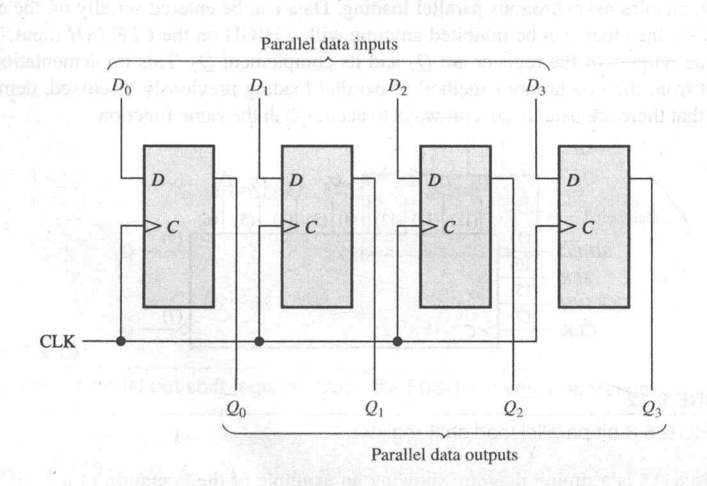
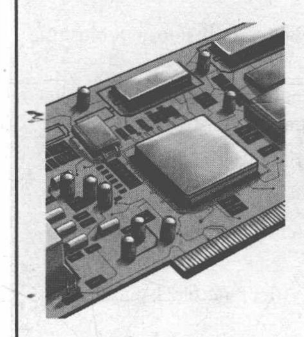


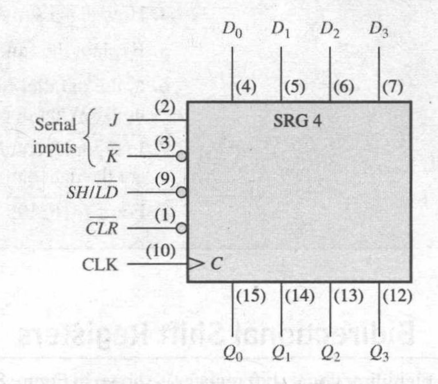
FIGURE 8-14

A parallel in/parallel out register.

IMPLEMENTATION: 4-BIT PARALLEL-ACCESS SHIFT REGISTER



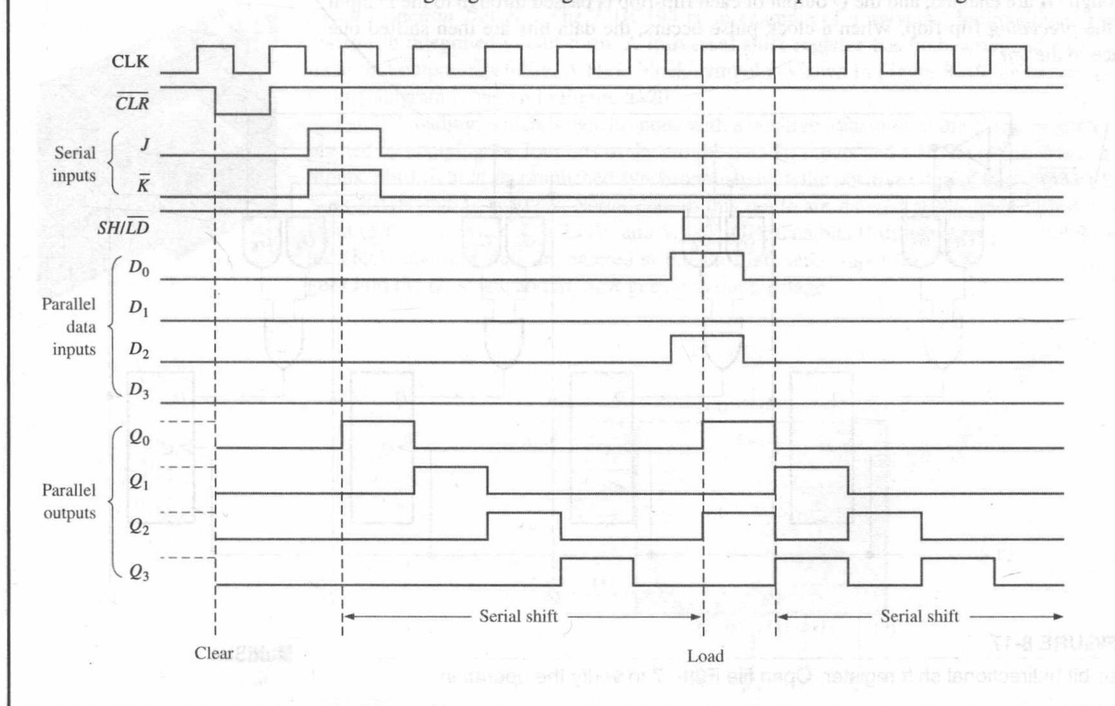
Fixed-Function Device The 74HC195 can be used for parallel in/parallel out operation. Because it also has a serial input, it can be used for serial in/serial out and serial in/parallel out operations. It can be used for parallel in/serial out operation by using Q_3 as the output. A typical logic block symbol is shown in Figure 8–15.



▶ **FIGURE 8-15**
The 74HC195 4-bit parallel access shift register.

When the $SHIFT/\overline{LOAD}$ input ($SH/\overline{LD}$) is LOW, the data on the parallel inputs are entered synchronously on the positive transition of the clock. When ($SH/\overline{LD}$) is HIGH, stored data will shift right (Q_0 to Q_3) synchronously with the clock. Inputs J and $\overline{K}$ are the serial data inputs to the first stage of the register (Q_0); Q_3 can be used for serial output data. The active-LOW clear input is asynchronous.

The timing diagram in Figure 8–16 illustrates the operation of this register.



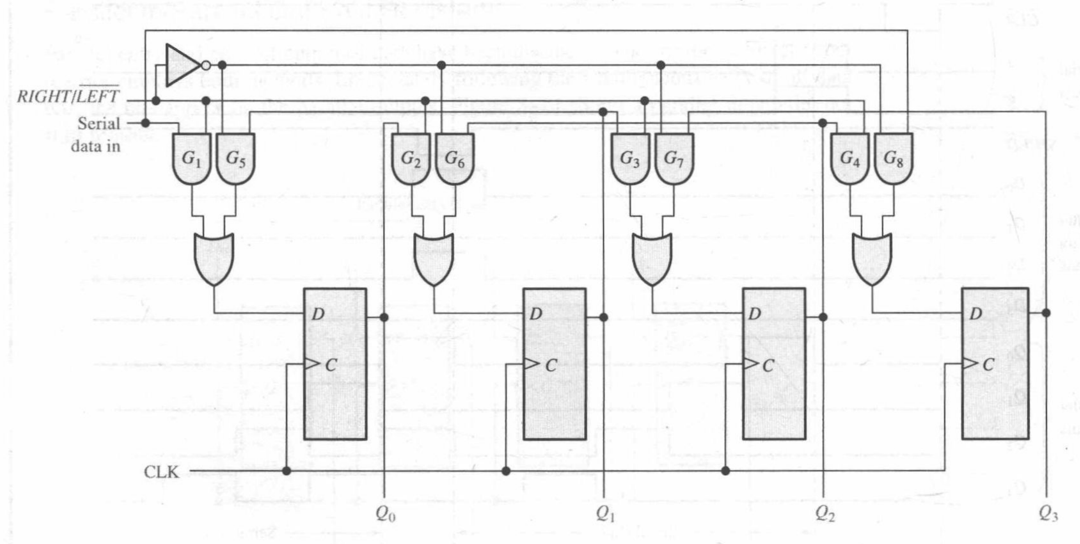
▲ **FIGURE 8-16**
Sample timing diagram for a 74HC195 shift register.

SECTION 8-2 CHECKUP

1. Develop the logic diagram for the shift register in Figure 8–3, using J-K flip-flops to replace the D flip-flops.
2. How many clock pulses are required to enter a byte of data serially into an 8-bit shift register?
3. The bit sequence 1101 is serially entered (least-significant bit first) into a 4-bit parallel out shift register that is initially clear. What are the Q outputs after two clock pulses?
4. How can a serial in/parallel out register be used as a serial in/serial out register?
5. Explain the function of the $SHIFT/\overline{LOAD}$ input.
6. Is the parallel load operation in a 74HC165 shift register synchronous or asynchronous? What does this mean?
7. In Figure 8–14, $D_0 = 1, D_1 = 0, D_2 = 0$, and $D_3 = 1$. After three clock pulses, what are the data outputs?
8. For a 74HC195, $SH/\overline{LD} = 1, J = 1$, and $\overline{K} = 1$. What is Q_0 after one clock pulse?

8-3 Bidirectional Shift Registers

A 4-bit bidirectional shift register is shown in Figure 8–17. A HIGH on the $RIGHT/\overline{LEFT}$ control input allows data bits inside the register to be shifted to the right, and a LOW enables data bits inside the register to be shifted to the left. An examination of the gating logic will make the operation apparent. When the $RIGHT/\overline{LEFT}$ control input is HIGH, gates G_1 through G_4 are enabled, and the state of the Q output of each flip-flop is passed through to the D input of the *following* flip-flop. When a clock pulse occurs, the data bits are shifted one place to the *right*. When the $RIGHT/\overline{LEFT}$ control input is LOW, gates G_5 through G_8 are enabled, and the Q output of each flip-flop is passed through to the D input of the *preceding* flip-flop. When a clock pulse occurs, the data bits are then shifted one place to the *left*.



▲ **FIGURE 8-17**

Four-bit bidirectional shift register. Open file F08-17 to verify the operation.

MultiSim

EXAMPLE 8-4

Determine the state of the shift register of Figure 8–17 after each clock pulse for the given $RIGHT/\overline{LEFT}$ control input waveform in Figure 8–18(a). Assume that $Q_0 = 1$, $Q_1 = 1, Q_2 = 0$, and $Q_3 = 1$ and that the serial data-input line is LOW.

▶ **FIGURE 8-18**

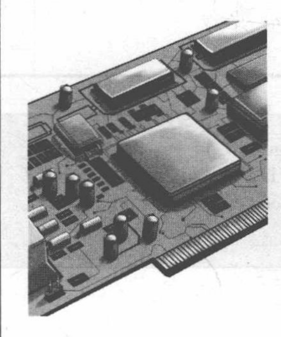
Solution

See Figure 8–18(b).

Related Problem

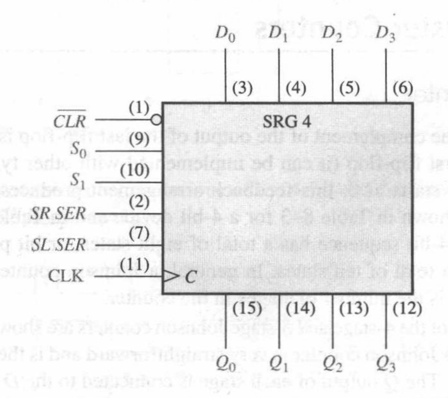
Invert the $RIGHT/\overline{LEFT}$ waveform, and determine the state of the shift register in Figure 8–17 after each clock pulse.

IMPLEMENTATION: 4-BIT BIDIRECTIONAL UNIVERSAL SHIFT REGISTER



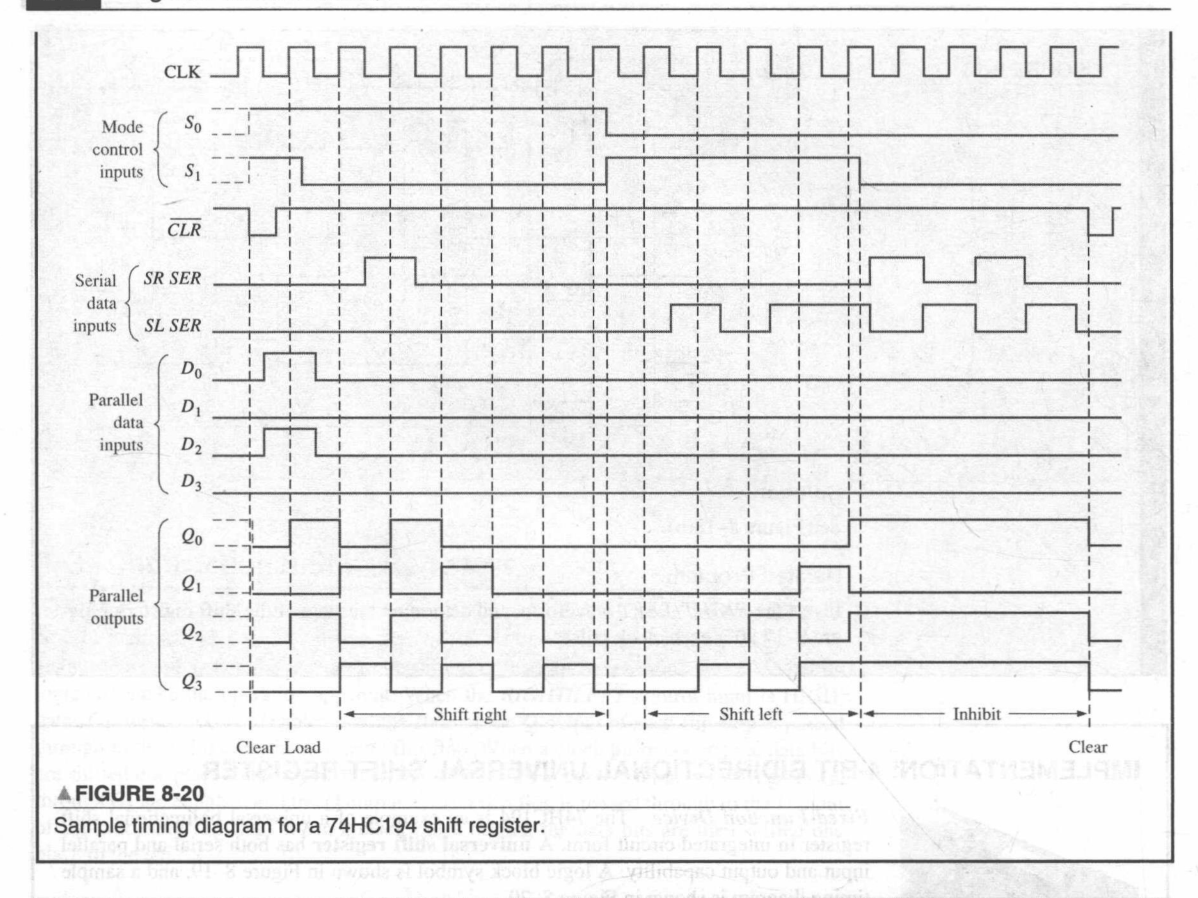
Fixed-Function Device The 74HC194 is an example of a universal bidirectional shift register in integrated circuit form. A **universal shift register** has both serial and parallel input and output capability. A logic block symbol is shown in Figure 8–19, and a sample timing diagram is shown in Figure 8–20.

Parallel loading, which is synchronous with a positive transition of the clock, is accomplished by applying the four bits of data to the parallel inputs and a HIGH to the S_0 and S_1 inputs. Shift right is accomplished synchronously with the positive edge of the clock when S_0 is HIGH and S_1 is LOW. Serial data in this mode are entered at the shift-right serial input (*SR SER*). When S_0 is LOW and S_1 is HIGH, data bits shift left synchronously with the clock, and new data are entered at the shift-left serial input (*SL SER*). Input *SR SER* goes into the Q_0 stage, and *SL SER* goes into the Q_3 stage.



▲ **FIGURE 8-19**

The 74HC194 4-bit bidirectional universal shift register.



▲ **FIGURE 8-20**

Sample timing diagram for a 74HC194 shift register.

SECTION 8-3 CHECKUP

1. Assume that the 4-bit bidirectional shift register in Figure 8–17 has the following contents: $Q_0 = 1, Q_1 = 1, Q_2 = 0$, and $Q_3 = 0$. There is a 1 on the serial data-input line. If *RIGHT*/$\overline{LEFT}$ is HIGH for three clock pulses and LOW for two more clock pulses, what are the contents after the fifth clock pulse?

8-4 Shift Register Counters

The Johnson Counter

In a **Johnson counter** the complement of the output of the last flip-flop is connected back to the *D* input of the first flip-flop (it can be implemented with other types of flip-flops as well). If the counter starts at 0, this feedback arrangement produces a characteristic sequence of states, as shown in Table 8–3 for a 4-bit device and in Table 8–4 for a 5-bit device. Notice that the 4-bit sequence has a total of eight states, or bit patterns, and that the 5-bit sequence has a total of ten states. In general, a Johnson counter will produce a modulus of $2n$, where n is the number of stages in the counter.

The implementations of the 4-stage and 5-stage Johnson counters are shown in Figure 8–21. The implementation of a Johnson counter is very straightforward and is the same regardless of the number of stages. The *Q* output of each stage is connected to the *D* input of the next stage (assuming that *D* flip-flops are used). The single exception is that the $\overline{Q}$ output of the last stage is connected back to the *D* input of the first stage. As the sequences in Table 8–3 and 8–4 show, if the counter starts at 0, it will "fill up" with 1s from left to right, and then it will "fill up" with 0s again.

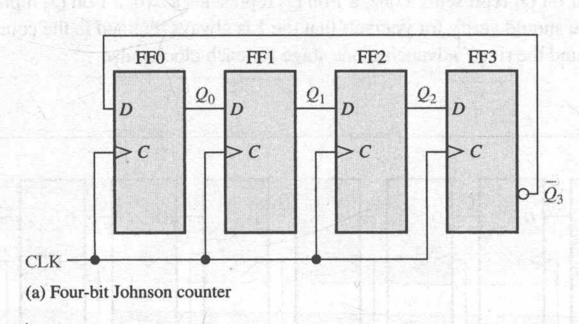
TABLE 8-3
Four-bit Johnson sequence.

Clock Pulse	Q_0	Q_1	Q_2	Q_3
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1

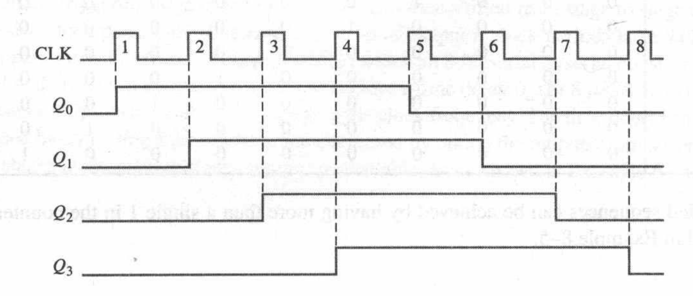
TABLE 8-4
Five-bit Johnson sequence.

Clock Pulse	Q_0	Q_1	Q_2	Q_3	Q_4
0	0	0	0	0	0
1	1	0	0	0	0
2	1	1	0	0	0
3	1	1	1	0	0
4	1	1	1	1	0
5	1	1	1	1	1
6	0	1	1	1	1
7	0	0	1	1	1
8	0	0	0	1	1
9	0	0	0	0	1

FIGURE 8-21
Four-bit and 5-bit Johnson counters.



(a) Four-bit Johnson counter



(b) Five-bit Johnson counter

Diagrams of the timing operations of the 4-bit and 5-bit counters are shown in Figures 8–22 and 8–23, respectively.

FIGURE 8-22
Timing sequence for a 4-bit Johnson counter.

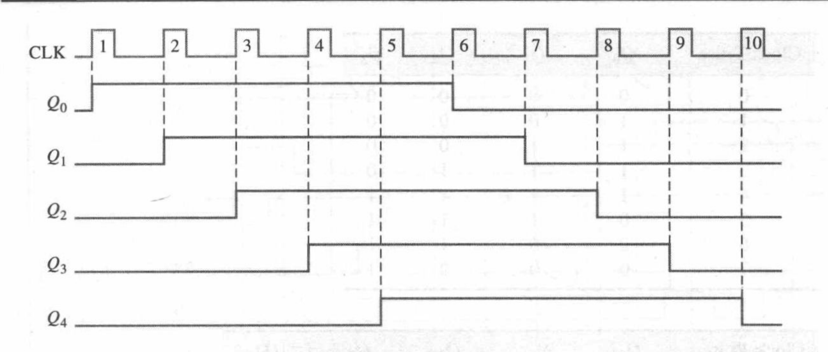
CLK 1 2 3 4 5 6 7 8

Q_0

Q_1

Q_2

Q_3



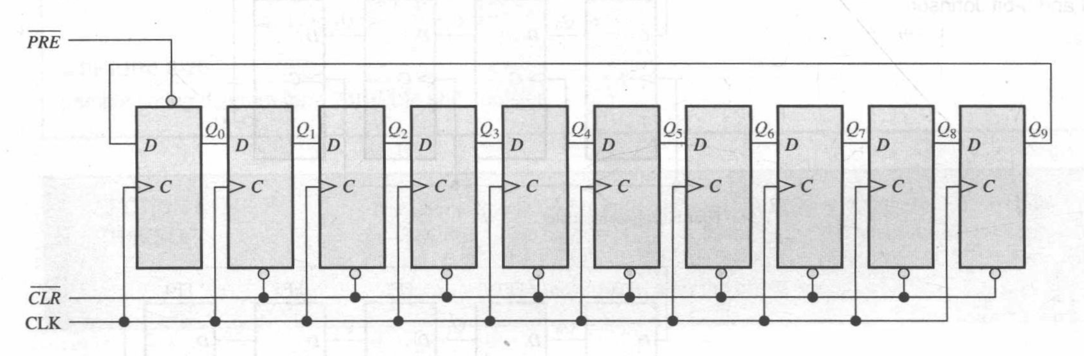
◀ **FIGURE 8-23**

Timing sequence for a 5-bit Johnson counter.

The Ring Counter

A **ring counter** utilizes one flip-flop for each state in its sequence. It has the advantage that decoding gates are not required. In the case of a 10-bit ring counter, there is a unique output for each decimal digit.

A logic diagram for a 10-bit ring counter is shown in Figure 8–24. The sequence for this ring counter is given in Table 8–5. Initially, a 1 is preset into the first flip-flop, and the rest of the flip-flops are cleared. Notice that the interstage connections are the same as those for a Johnson counter, except that Q rather than $\overline{Q}$ is fed back from the last stage. The ten outputs of the counter indicate directly the decimal count of the clock pulse. For instance, a 1 on Q_0 represents a zero, a 1 on Q_1 represents a one, a 1 on Q_2 represents a two, a 1 on Q_3 represents a three, and so on. You should verify for yourself that the 1 is always retained in the counter and simply shifted "around the ring," advancing one stage for each clock pulse.



▲ **FIGURE 8-24**

A 10-bit ring counter. Open file F08-24 to verify operation.

MultiSim

Clock Pulse	Q_0	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9
0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0
4	0	0	0	0	1	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0	0
6	0	0	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	0	1	0	0
8	0	0	0	0	0	0	0	0	1	0
9	0	0	0	0	0	0	0	0	0	1

◀ **TABLE 8-5**

Ten-bit ring counter sequence.

Modified sequences can be achieved by having more than a single 1 in the counter, as illustrated in Example 8–5.

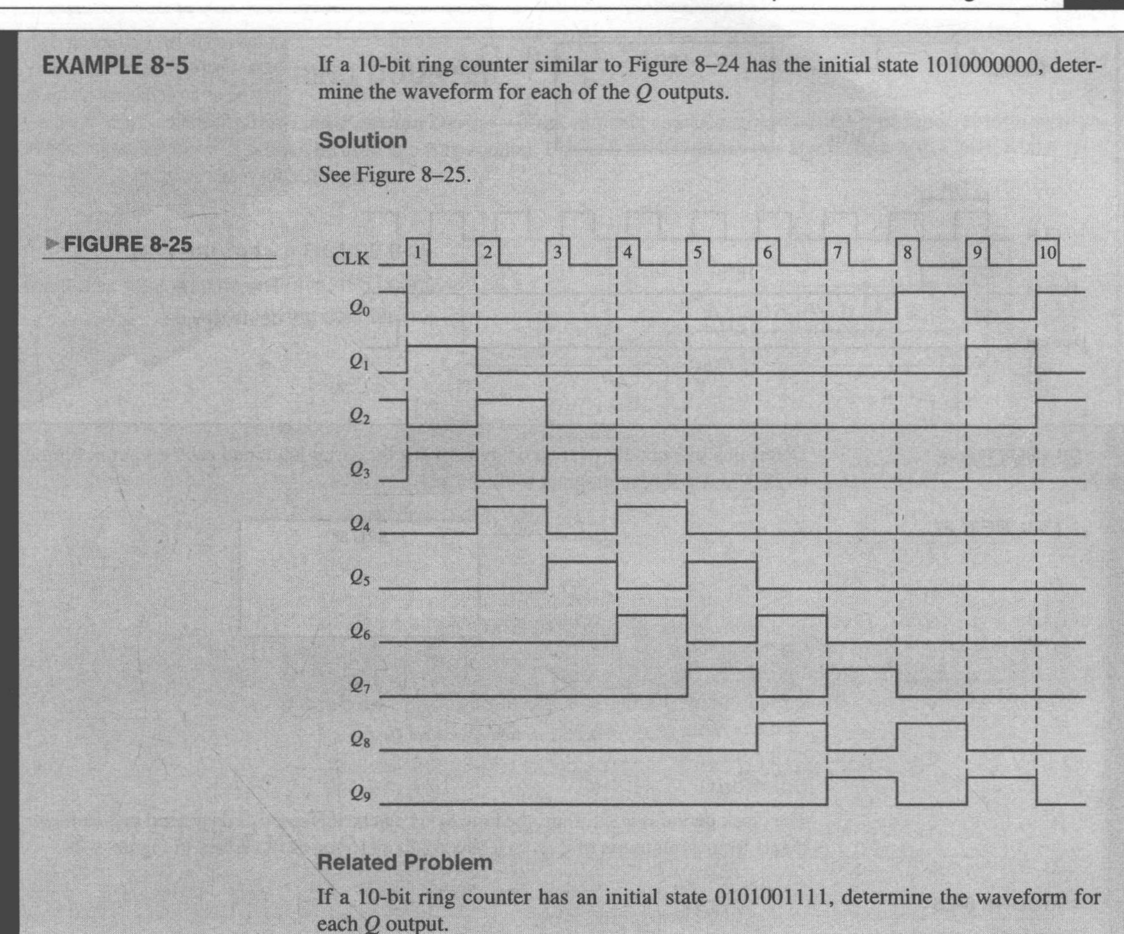
EXAMPLE 8-5

If a 10-bit ring counter similar to Figure 8–24 has the initial state 1010000000, determine the waveform for each of the Q outputs.

Solution

See Figure 8–25.

► **FIGURE 8-25**



Related Problem

If a 10-bit ring counter has an initial state 0101001111, determine the waveform for each Q output.

SECTION 8-4 CHECKUP

1. How many states are there in an 8-bit Johnson counter sequence?
2. Write the sequence of states for a 3-bit Johnson counter starting with 000.

8-5 Shift Register Applications

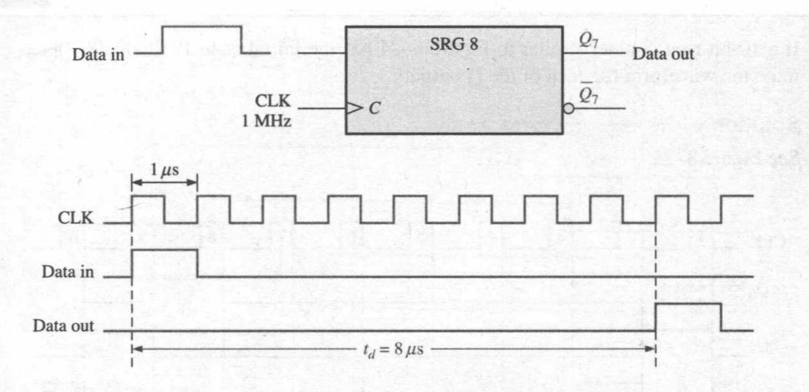
InfoNote

Microprocessors have special instructions that can emulate a serial shift register. The accumulator register can shift data to the left or right. A right shift is equivalent to a divide-by-2 operation and a left shift is equivalent to a multiply-by-2 operation. Data in the accumulator can be shifted left or right with the rotate instructions; ROR is the rotate right instruction, and ROL

Time Delay

A serial in/serial out shift register can be used to provide a time delay from input to output that is a function of both the number of stages (n) in the register and the clock frequency.

When a data pulse is applied to the serial input as shown in Figure 8–26, it enters the first stage on the triggering edge of the clock pulse. It is then shifted from stage to stage on each successive clock pulse until it appears on the serial output n clock periods later. This time-delay operation is illustrated in Figure 8–26, in which an 8-bit serial in/serial out shift register is used with a clock frequency of 1 MHz to achieve a time delay (t_d) of 8 μs (8 $\times$ 1 μs). This time can be adjusted up or down by changing the clock frequency. The time delay can also be increased by cascading shift registers and decreased by taking the outputs from successively lower stages in the register if the outputs are available, as illustrated in Example 8–6.



is the rotate left instruction. Two other instructions treat the carry flag bit as an additional bit for the rotate operation. These are the RCR for rotate carry right and RCL for rotate carry left.

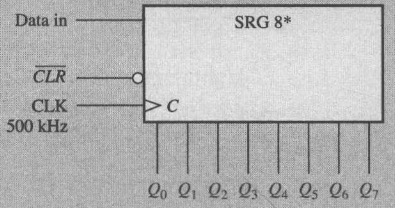
◀ **FIGURE 8-26**

The shift register as a time-delay device.

EXAMPLE 8-6

Determine the amount of time delay between the serial input and each output in Figure 8–27. Show a timing diagram to illustrate.

▶ **FIGURE 8-27**

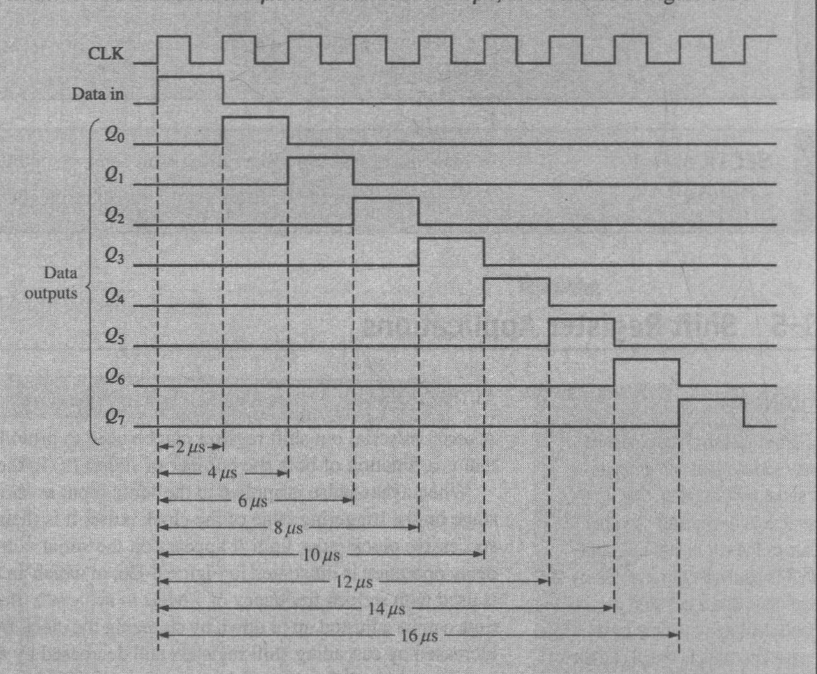


Solution

The clock period is 2 μs. Thus, the time delay can be increased or decreased in 2 μs increments from a minimum of 2 μs to a maximum of 16 μs, as illustrated in Figure 8–28.

▶ **FIGURE 8-28**

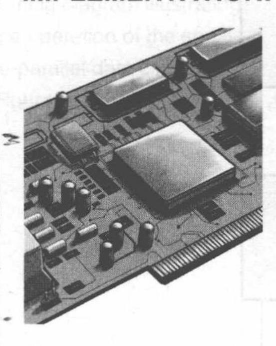
Timing diagram showing time delays for the register in Figure 8–27.



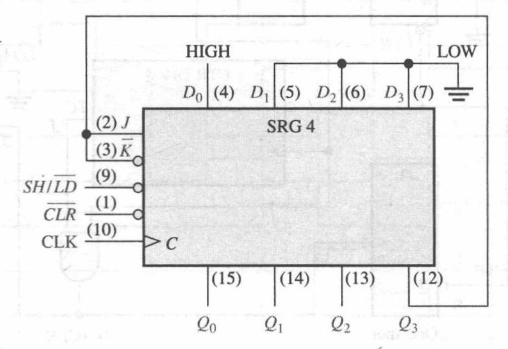
Related Problem

Determine the clock frequency required to obtain a time delay of 24 μs to the Q_7 output in Figure 8–27.

IMPLEMENTATION: A RING COUNTER



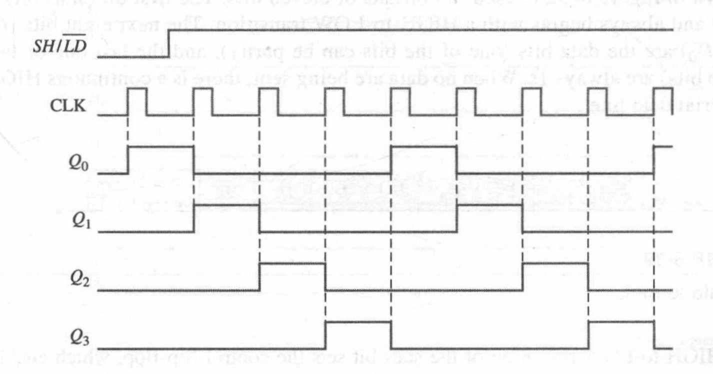
Fixed-Function Device If the output is connected back to the serial input, a shift register can be used as a ring counter. Figure 8–29 illustrates this application with a 74HC195 4-bit shift register.



▶ **FIGURE 8-29**

74HC195 connected as a ring counter.

Initially, a bit pattern of 1000 (or any other pattern) can be synchronously preset into the counter by applying the bit pattern to the parallel data inputs, taking the $SH/\overline{LD}$ input LOW, and applying a clock pulse. After this initialization, the 1 continues to circulate through the ring counter, as the timing diagram in Figure 8–30 shows.



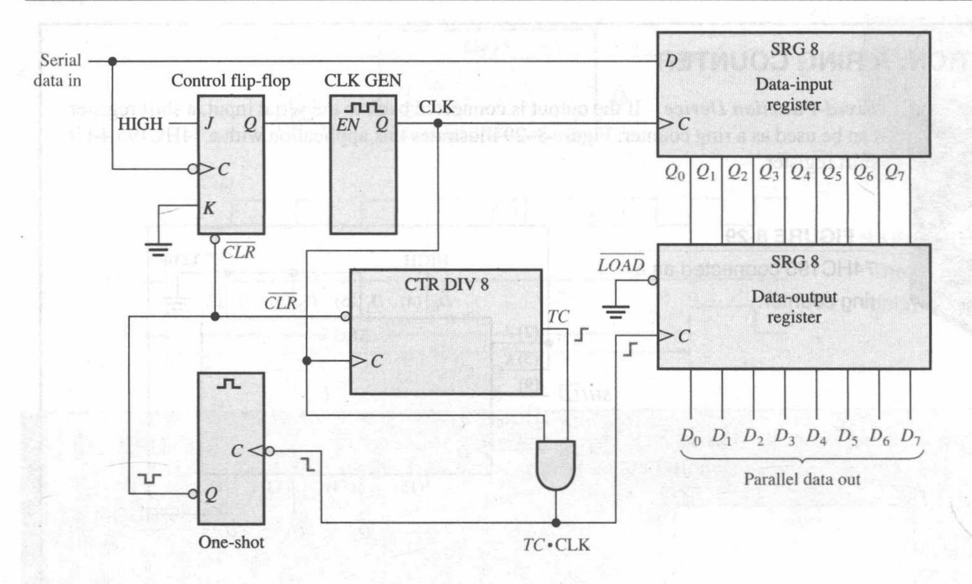
▲ **FIGURE 8-30**

Timing diagram showing two complete cycles of the ring counter in Figure 8–29 when it is initially preset to 1000.

Serial-to-Parallel Data Converter

Serial data transmission from one digital system to another is commonly used to reduce the number of wires in the transmission line. For example, eight bits can be sent serially over one wire, but it takes eight wires to send the same data in parallel.

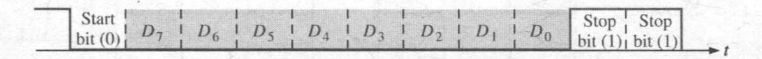
Serial data transmission is widely used by peripherals to pass data back and forth to a computer. For example, USB (universal serial bus) is used to connect keyboards printers, scanners, and more to the computer. All computers process data in parallel form, thus the requirement for serial-to-parallel conversion. A simplified serial-to-parallel data converter, in which two types of shift registers are used, is shown in Figure 8–31.



▲ **FIGURE 8-31**

Simplified logic diagram of a serial-to-parallel converter.

To illustrate the operation of this serial-to-parallel converter, the serial data format shown in Figure 8–32 is used. It consists of eleven bits. The first bit (start bit) is always 0 and always begins with a HIGH-to-LOW transition. The next eight bits (D_7 through D_0) are the data bits (one of the bits can be parity), and the last one or two bits (stop bits) are always 1s. When no data are being sent, there is a continuous HIGH on the serial data line.



▲ **FIGURE 8-32**

Serial data format.

The HIGH-to-LOW transition of the start bit sets the control flip-flop, which enables the clock generator. After a fixed delay time, the clock generator begins producing a pulse waveform, which is applied to the data-input register and to the divide-by-8 counter. The clock has a frequency precisely equal to that of the incoming serial data, and the first clock pulse after the start bit occurs during the first data bit.

The timing diagram in Figure 8–33 illustrates the following basic operation: The eight data bits (D_7 through D_0) are serially shifted into the data-input register. Shortly after the eighth clock pulse, the terminal count (*TC*) goes from LOW to HIGH, indicating the counter is at the last state. This rising edge is ANDed with the clock pulse, which is still HIGH, producing a rising edge at $TC \cdot$ CLK. This parallel loads the eight data bits from the data-input shift register to the data-output register. A short time later, the clock pulse goes LOW and this HIGH-to-LOW transition triggers the one-shot, which produces a short-duration pulse to clear the counter and reset the control flip-flop and thus disable the clock generator. The system is now ready for the next group of eleven bits, and it waits for the next HIGH-to-LOW transition at the beginning of the start bit.

By reversing the process just stated, parallel-to-serial data conversion can be accomplished. Since the serial data format must be produced, start and stop bits must be added to the sequence.

FIGURE 8-33

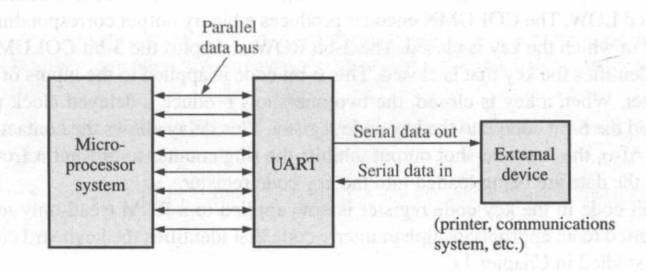
Timing diagram illustrating the operation of the serial-to-parallel data converter in Figure 8–31.

Universal Asynchronous Receiver Transmitter (UART)

As mentioned, computers and microprocessor-based systems often send and receive data in a parallel format. Frequently, these systems must communicate with external devices that send and/or receive serial data. An interfacing device used to accomplish these conversions is the UART (Universal Asynchronous Receiver Transmitter). Figure 8–34 illustrates the UART in a general microprocessor-based system application.

FIGURE 8-34

UART interface.



A UART includes both serial-to-parallel and parallel-to-serial conversion, as shown in the block diagram in Figure 8–35. The data bus is basically a set of parallel conductors along which data move between the UART and the microprocessor system. Buffers interface the data registers with the data bus.

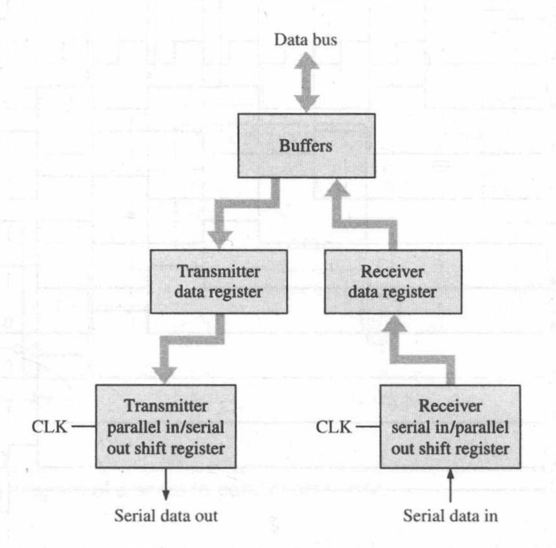


FIGURE 8-35
Basic UART block diagram.

The UART receives data in serial format, converts the data to parallel format, and places them on the data bus. The UART also accepts parallel data from the data bus, converts the data to serial format, and transmits them to an external device.

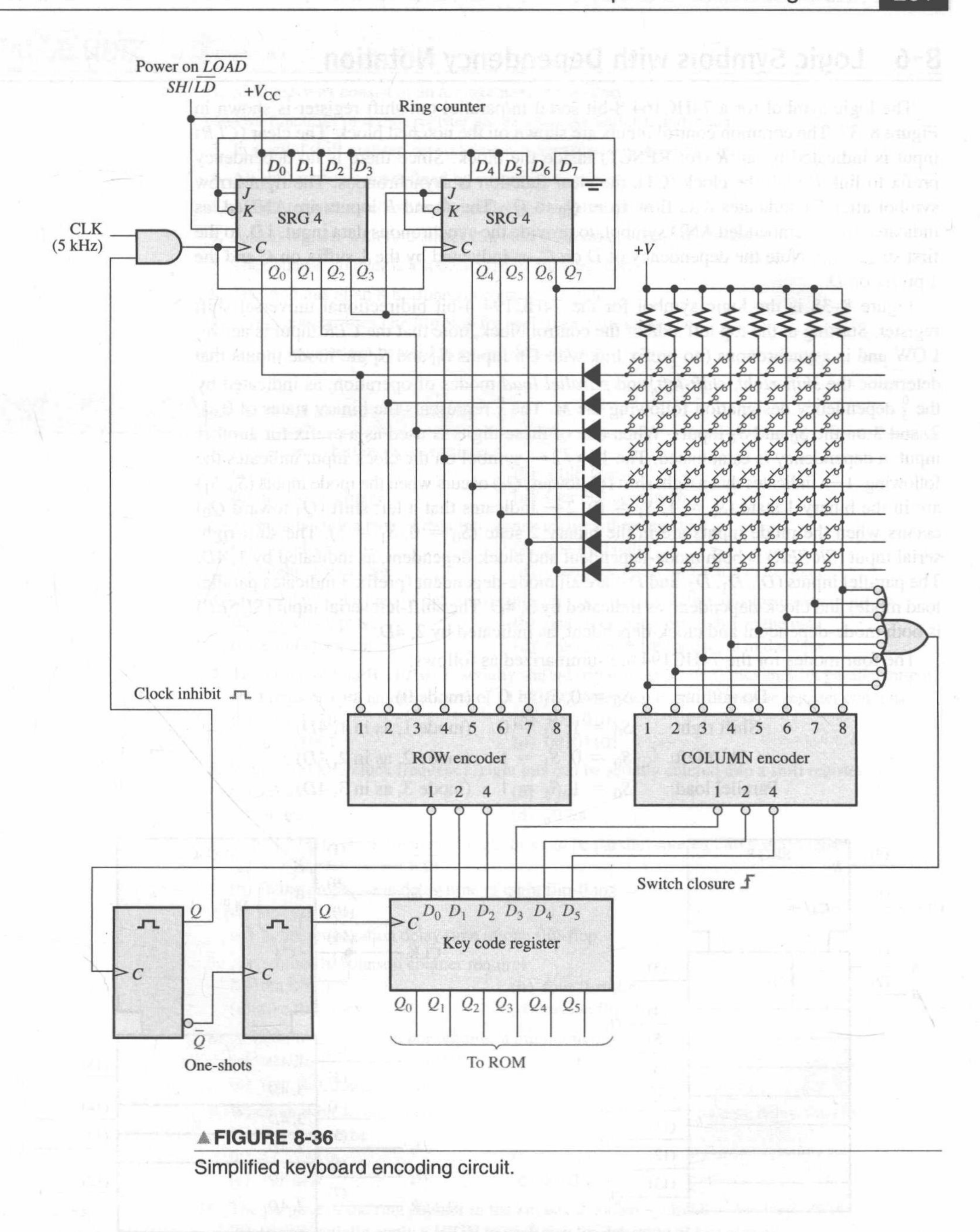
Keyboard Encoder

The keyboard encoder is a good example of the application of a shift register used as a ring counter in conjunction with other devices. Recall that a simplified computer keyboard encoder without data storage was presented in Chapter 6.

Figure 8–36 shows a simplified keyboard encoder for encoding a key closure in a 64-key matrix organized in eight rows and eight columns. Two parallel in/parallel out 4-bit shift registers are connected as an 8-bit ring counter with a fixed bit pattern of seven 1s and one 0 preset into it when the power is turned on. Two priority encoders (introduced in Chapter 6) are used as eight-line-to-three-line encoders (9 input HIGH, 8 output unused) to encode the ROW and COLUMN lines of the keyboard matrix. A parallel in/parallel out register (key code) stores the ROW/COLUMN code from the priority encoders.

The basic operation of the keyboard encoder in Figure 8–36 is as follows: The ring counter "scans" the rows for a key closure as the clock signal shifts the 0 around the counter at a 5 kHz rate. The 0 (LOW) is sequentially applied to each ROW line, while all other ROW lines are HIGH. All the ROW lines are connected to the ROW encoder inputs, so the 3-bit output of the ROW encoder at any time is the binary representation of the ROW line that is LOW. When there is a key closure, one COLUMN line is connected to one ROW line. When the ROW line is taken LOW by the ring counter, that particular COLUMN line is also pulled LOW. The COLUMN encoder produces a binary output corresponding to the COLUMN in which the key is closed. The 3-bit ROW code plus the 3-bit COLUMN code uniquely identifies the key that is closed. This 6-bit code is applied to the inputs of the key code register. When a key is closed, the two one-shots produce a delayed clock pulse to parallel-load the 6-bit code into the key code register. This delay allows the contact bounce to die out. Also, the first one-shot output inhibits the ring counter to prevent it from scanning while the data are being loaded into the key code register.

The 6-bit code in the key code register is now applied to a ROM (read-only memory) to be converted to an appropriate alphanumeric code that identifies the keyboard character. ROMs are studied in Chapter 11.



▲**FIGURE 8-36**

Simplified keyboard encoding circuit.

SECTION 8-5 CHECKUP

1. In the keyboard encoder, how many times per second does the ring counter scan the keyboard?
2. What is the 6-bit ROW/COLUMN code (key code) for the top row and the left-most column in the keyboard encoder?
3. What is the purpose of the diodes in the keyboard encoder? What is the purpose of the resistors?

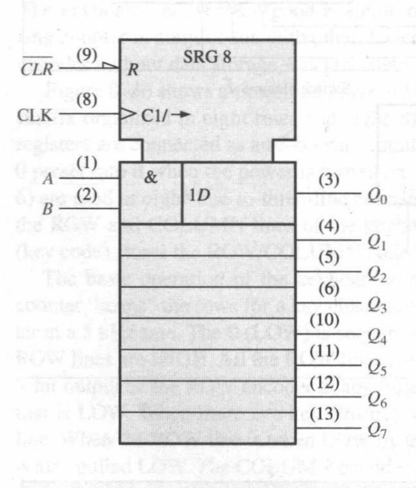
8-6 Logic Symbols with Dependency Notation

The logic symbol for a 74HC164 8-bit serial in/parallel out shift register is shown in Figure 8–37. The common control inputs are shown on the notched block. The clear ($\overline{CLR}$) input is indicated by an R (for RESET) inside the block. Since there is no dependency prefix to link R with the clock (C1), the clear function is asynchronous. The right arrow symbol after C1 indicates data flow from Q_0 to Q_7. The A and B inputs are ANDed, as indicated by the embedded AND symbol, to provide the synchronous data input, 1D, to the first stage (Q_0). Note the dependency of D on C, as indicated by the 1 suffix on C and the 1 prefix on D.

Figure 8–38 is the logic symbol for the 74HC194 4-bit bidirectional universal shift register. Starting at the top left side of the control block, note that the $\overline{CLR}$ input is active-LOW and is asynchronous (no prefix link with C). Inputs S_0 and S_1 are mode inputs that determine the *shift-right, shift-left*, and *parallel load* modes of operation, as indicated by the $\frac{0}{3}$ dependency designation following the M. The $\frac{0}{3}$ represents the binary states of 0, 1, 2, and 3 on the S_0 and S_1 inputs. When one of these digits is used as a prefix for another input, a dependency is established. The $1 \rightarrow /2 \leftarrow$ symbol on the clock input indicates the following: $1 \rightarrow$ indicates that a right shift (Q_0 toward Q_3) occurs when the mode inputs (S_0, S_1) are in the binary 1 state ($S_0 = 1, S_1 = 0$), $2 \leftarrow$ indicates that a left shift (Q_3 toward Q_0) occurs when the mode inputs are in the binary 2 state ($S_0 = 0, S_1 = 1$). The shift-right serial input (*SR SER*) is both mode-dependent and clock-dependent, as indicated by 1, 4D. The parallel inputs (D_0, D_1, D_2, and D_3) are all mode-dependent (prefix 3 indicates parallel load mode) and clock-dependent, as indicated by 3, 4D. The shift-left serial input (*SL SER*) is both mode-dependent and clock-dependent, as indicated by 2, 4D.

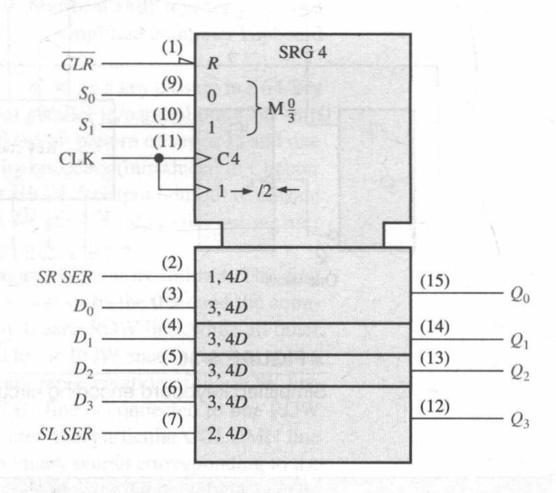
The four modes for the 74HC194 are summarized as follows:

Do nothing:	$S_0 = 0, S_1 = 0$	(mode 0)
Shift right:	$S_0 = 1, S_1 = 0$	(mode 1, as in 1, 4D)
Shift left:	$S_0 = 0, S_1 = 1$	(mode 2, as in 2, 4D)
Parallel load:	$S_0 = 1, S_1 = 1$	(mode 3, as in 3, 4D)



▲ **FIGURE 8-37**

Logic symbol for the 74HC164.



▲ **FIGURE 8-38**

Logic symbol for the 74HC194.

SECTION 8-6 CHECKUP

1. In Figure 8–38, are there any inputs that are dependent on the mode inputs being in the 0 state?
2. Is the parallel load synchronous with the clock?

TRUE/FALSE QUIZ

Answers are at the end of the chapter.

1. Shift registers consist of an arrangement of flip-flops.
2. Two functions of a shift register are data storage and data movement.
3. In a serial shift register, several data bits are entered at the same time.
4. All shift registers are defined by specified sequences.
5. A shift register can have both parallel and serial outputs.
6. A shift register with four stages can store a maximum count of fifteen.
7. The Johnson counter is a special type of shift register.
8. The modulus of an 8-bit Johnson counter is eight.
9. A ring counter uses one flip-flop for each state in its sequence.
10. A shift register can be used as a time delay device.

SELF-TEST

Answers are at the end of the chapter.

1. A stage in a shift register consists of
   (a) a latch **(b)** a flip-flop
   (c) a byte of storage **(d)** four bits of storage
2. To serially shift a byte of data into a shift register, there must be
   (a) one clock pulse **(b)** one load pulse
   (c) eight clock pulses **(d)** one clock pulse for each 1 in the data
3. To parallel load a byte of data into a shift register with a synchronous load, there must be
   (a) one clock pulse **(b)** one clock pulse for each 1 in the data
   (c) eight clock pulses **(d)** one clock pulse for each 0 in the data
4. The group of bits 10110101 is serially shifted (right-most bit first) into an 8-bit parallel output shift register with an initial state of 11100100. After two clock pulses, the register contains
   (a) 01011110 **(b)** 10110101
   (c) 01111001 **(d)** 00101101
5. With a 100 kHz clock frequency, eight bits can be serially entered into a shift register in
   (a) 80 μs **(b)** 8 μs
   (c) 80 ms **(d)** 10 μs
6. With a 1 MHz clock frequency, eight bits can be parallel entered into a shift register
   (a) in 8 μs
   (b) in the propagation delay time of eight flip-flops
   (c) in 1 μs
   (d) in the propagation delay time of one flip-flop
7. A modulus-10 Johnson counter requires
   (a) ten flip-flops **(b)** four flip-flops
   (c) five flip-flops **(d)** twelve flip-flops
8. A modulus-10 ring counter requires a minimum of
   (a) ten flip-flops **(b)** five flip-flops
   (c) four flip-flops **(d)** twelve flip-flops
9. When an 8-bit serial in/serial out shift register is used for a 24 μs time delay, the clock frequency must be
   (a) 41.67 kHz **(b)** 333 kHz
   (c) 125 kHz **(d)** 8 MHz
10. The purpose of the ring counter in the keyboard encoding circuit of Figure 8–36 is
    (a) to sequentially apply a HIGH to each row for detection of key closure
    (b) to provide trigger pulses for the key code register
    (c) to sequentially apply a LOW to each row for detection of key closure
    (d) to sequentially reverse bias the diodes in each row

PROBLEMS

Answers to odd-numbered problems are at the end of the book.

Section 8–1 Shift Register Operations

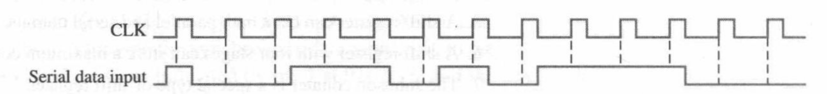
1. Why are shift registers considered basic memory devices?
2. What is the storage capacity of a register that can retain two bytes of data?
3. Name two functions of a shift register.

Section 8–2 Types of Shift Register Data I/Os

4. The sequence 1011 is applied to the input of a 4-bit serial shift register that is initially cleared. What is the state of the shift register after three clock pulses?

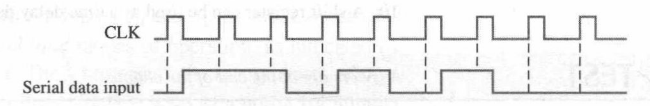
5. For the data input and clock in Figure 8–39, determine the states of each flip-flop in the shift register of Figure 8–3 and show the Q waveforms. Assume that the register contains all 1s initially.

FIGURE 8-39



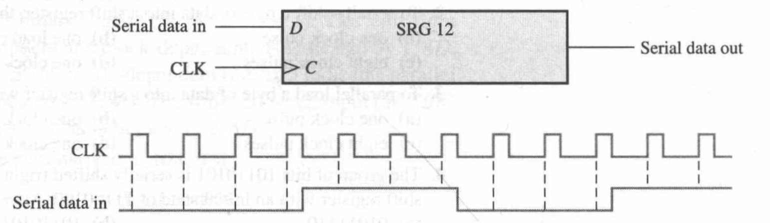
6. Solve Problem 5 for the waveforms in Figure 8–40.

FIGURE 8-40

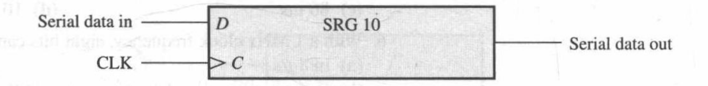


7. What is the state of the register in Figure 8–41 after each clock pulse if it starts in the 101001111000 state?

FIGURE 8-41



8. For the serial in/serial out shift register, determine the data-output waveform for the data-input and clock waveforms in Figure 8–42. Assume that the register is initially cleared.



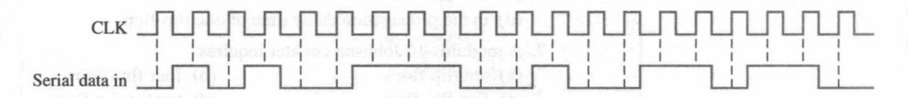
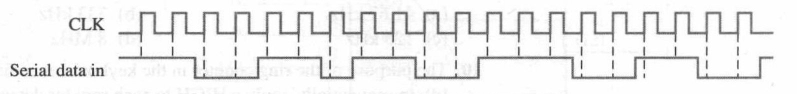


FIGURE 8-42

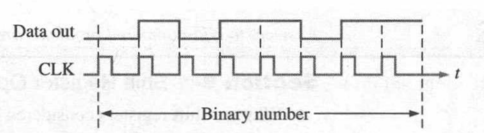
9. Solve Problem 8 for the waveforms in Figure 8–43.

FIGURE 8-43



10. A leading-edge clocked serial in/serial out shift register has a data-output waveform as shown in Figure 8–44. What binary number is stored in the 8-bit register if the first data bit out (left-most) is the LSB?

FIGURE 8-44

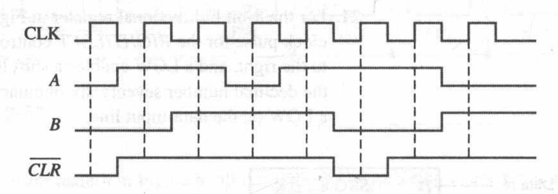


11. Show a complete timing diagram including the parallel outputs for the shift register in Figure 8–6. Use the waveforms in Figure 8–42 with the register initially clear.

12. Solve Problem 11 for the input waveforms in Figure 8–43.

13. Develop the Q_0 through Q_7 outputs for a 74HC164 shift register with the input waveforms shown in Figure 8–45.

FIGURE 8-45



14. The shift register in Figure 8–46(a) has $SHIFT/\overline{LOAD}$ and CLK inputs as shown in part (b). The serial data input (*SER*) is a 0. The parallel data inputs are $D_0 = 1$, $D_1 = 0$, $D_2 = 1$, and $D_3 = 0$ as shown. Develop the data-output waveform in relation to the inputs.

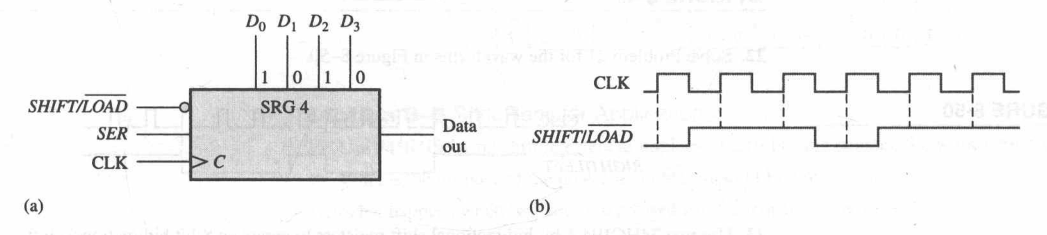
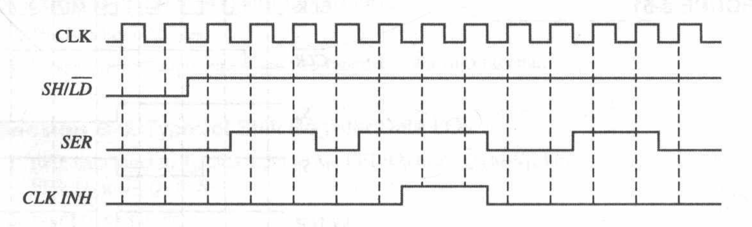


FIGURE 8-46

15. The waveforms in Figure 8–47 are applied to a 74HC165 shift register. The parallel inputs are all 0. Determine the Q_7 waveform.

FIGURE 8-47

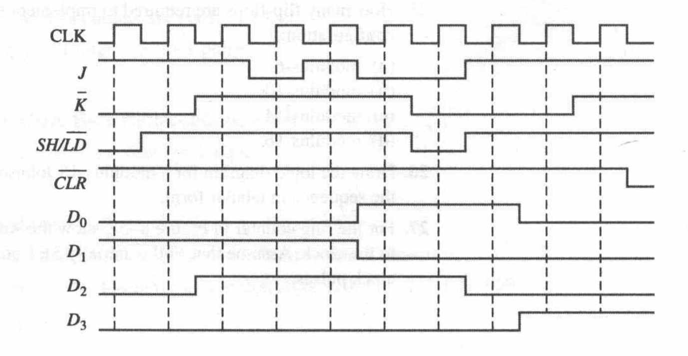


16. Solve Problem 15 if the parallel inputs are all 1.

17. Solve Problem 15 if the *SER* input is inverted.

18. Determine all the Q output waveforms for a 74HC195 4-bit shift register when the inputs are as shown in Figure 8–48.

FIGURE 8-48

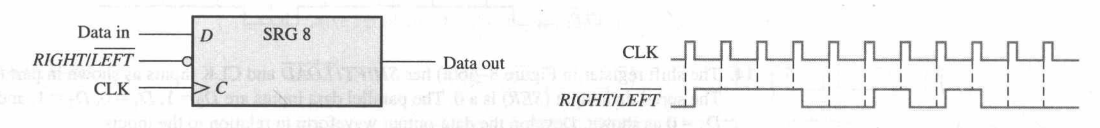


19. Solve Problem 18 if the $SH/\overline{LD}$ input is inverted and the register is initially clear.

20. Use two 74HC195 shift registers to form an 8-bit shift register. Show the required connections.

Section 8–3 Bidirectional Shift Registers

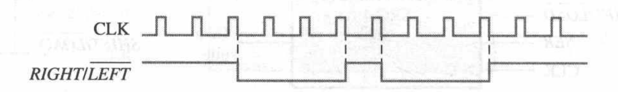
21. For the 8-bit bidirectional register in Figure 8–49, determine the state of the register after each clock pulse for the $RIGHT/\overline{LEFT}$ control waveform given. A HIGH on this input enables a shift to the right, and a LOW enables a shift to the left. Assume that the register is initially storing the decimal number seventy-six in binary, with the right-most position being the LSB. There is a LOW on the data-input line.



▲ FIGURE 8-49

22. Solve Problem 21 for the waveforms in Figure 8–50.

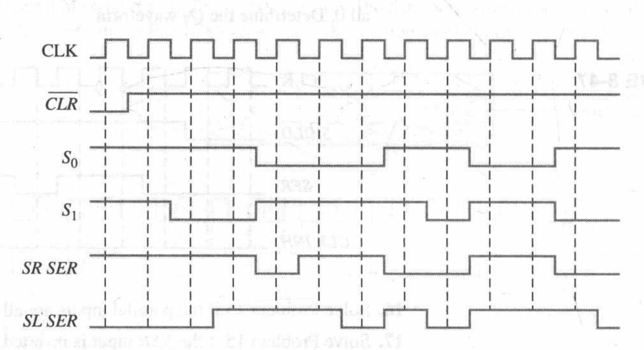
▶ FIGURE 8-50



23. Use two 74HC194 4-bit bidirectional shift registers to create an 8-bit bidirectional shift register. Show the connections.

24. Determine the Q outputs of a 74HC194 with the inputs shown in Figure 8–51. Inputs D_0, D_1, D_2, and D_3 are all HIGH.

▶ FIGURE 8-51



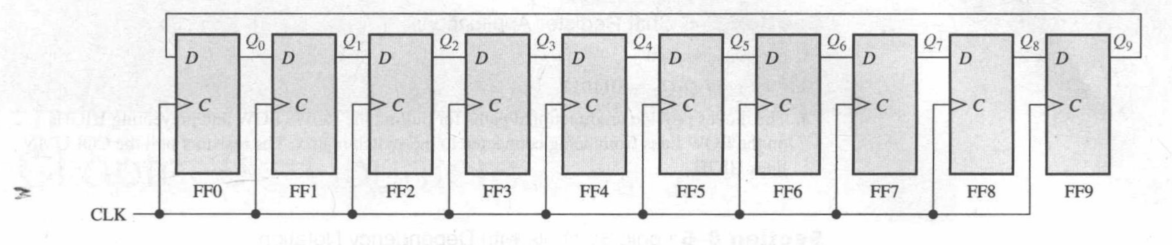
Section 8–4 Shift Register Counters

25. How many flip-flops are required to implement each of the following in a Johnson counter configuration:

(a) modulus-6
(b) modulus-10
(c) modulus-14
(d) modulus-16

26. Draw the logic diagram for a modulus-18 Johnson counter. Show the timing diagram and write the sequence in tabular form.

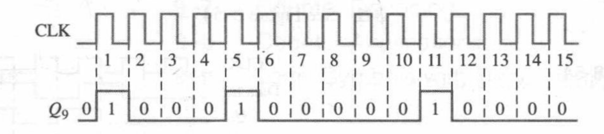
27. For the ring counter in Figure 8–52, show the waveforms for each flip-flop output with respect to the clock. Assume that FF0 is initially SET and that the rest are RESET. Show at least ten clock pulses.



▲ FIGURE 8-52

28. The waveform pattern in Figure 8–53 is required. Devise a ring counter, and indicate how it can be preset to produce this waveform on its Q_9 output. At CLK16 the pattern begins to repeat.

▶ FIGURE 8-53



Section 8–5 Shift Register Applications

29. Use 74HC195 4-bit shift registers to implement a 16-bit ring counter. Show the connections.
30. What is the purpose of the power-on $\overline{LOAD}$ input in Figure 8–36?
31. What happens when two keys are pressed simultaneously in Figure 8–36?

ANSWERS

SECTION CHECKUPS

Section 8–1 Shift Register Operations

1. The number of stages.
2. Storage and data movement are two functions of a shift register.

Section 8–2 Types of Shift Register Data I/Os

1. FF0: data input to J_0, $\overline{\text{data input}}$ to K_0; FF1: Q_0 to J_1, $\overline{Q}_0$ to K_1; FF2: Q_1 to J_2, $\overline{Q}_1$ to K_2; FF3: Q_2 to J_3, $\overline{Q}_2$ to K_3
2. Eight clock pulses
3. 0100 after 2 clock pulses
4. Take the serial output from the right-most flip-flop for serial out operation.
5. When $SHIFT/\overline{LOAD}$ is HIGH, the data are shifted right one bit per clock pulse. When $SHIFT/\overline{LOAD}$ is LOW, the data on the parallel inputs are loaded into the register.
6. The parallel load operation is asynchronous, so it is not dependent on the clock.
7. The data outputs are 1001.
8. $Q_0 = 1$ after one clock pulse

Section 8–3 Bidirectional Shift Registers

1. 1111 after the fifth clock pulse

Section 8–4 Shift Register Counters

1. Sixteen states are in an 8-bit Johnson counter sequence.
2. For a 3-bit Johnson counter: 000, 100, 110, 111, 011, 001, 000

Section 8–5 Shift Register Applications

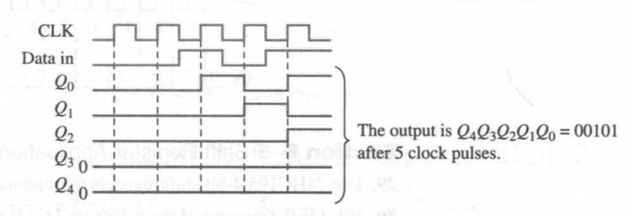
1. 625 scans/second
2. $Q_5Q_4Q_3Q_2Q_1Q_0 = 011011$
3. The diodes provide unidirectional paths for pulling the ROWs LOW and preventing HIGHs on the ROW lines from being connected to the switch matrix. The resistors pull the COLUMN lines HIGH.

Section 8–6 Logic Symbols with Dependency Notation

1. No inputs are dependent on the mode inputs being in the 0 state.
2. Yes, the parallel load is synchronous with the clock as indicated by the 4*D* label.

RELATED PROBLEMS FOR EXAMPLES

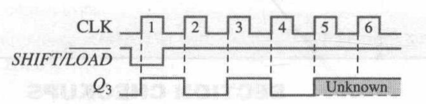
8–1 See Figure 8–54.



▶ **FIGURE 8-54**

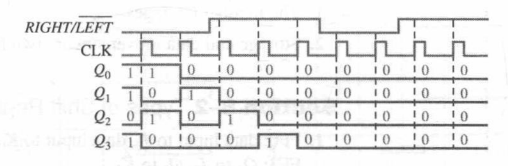
8–2 The state of the register after three additional clock pulses is 0000.

8–3 See Figure 8–55.



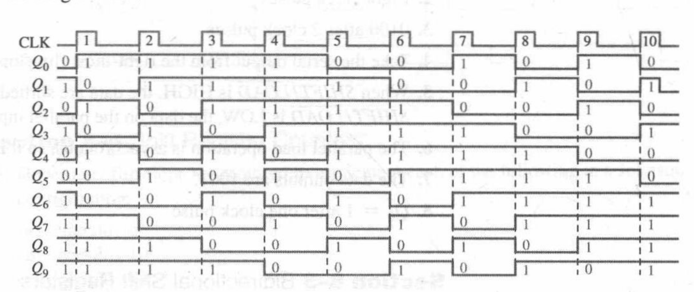
▶ **FIGURE 8-55**

8–4 See Figure 8–56.



▶ **FIGURE 8-56**

8–5 See Figure 8–57.



▶ **FIGURE 8-57**

8–6 $f = 1/3\ \mu s = 333$ kHz

TRUE/FALSE QUIZ

1. T **2.** T **3.** F **4.** F **5.** T **6.** T **7.** T **8.** F **9.** T **10.** T

SELF-TEST

1. (b) **2.** (c) **3.** (a) **4.** (c) **5.** (a) **6.** (d) **7.** (c) **8.** (a) **9.** (b) **10.** (c)

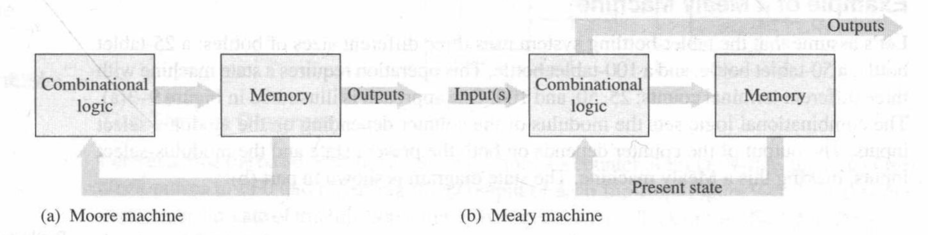
Chapter 9 Counters

CHAPTER OUTLINE

9-1 Finite State Machines

General Models of Finite State Machines

A Moore state machine consists of combinational logic that determines the sequence and memory (flip-flops), as shown in Figure 9–1(a). A Mealy state machine is shown in part (b).



▲FIGURE 9-1

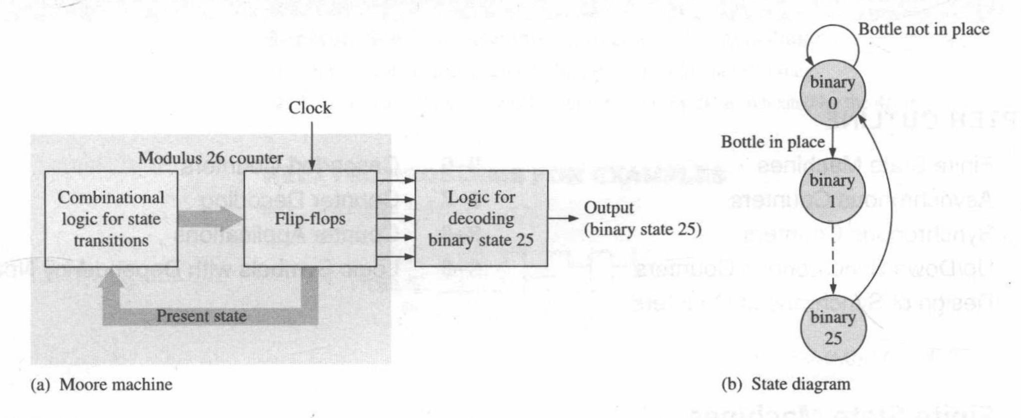
Two types of sequential logic.

In the Moore machine, the combinational logic is a gate array with outputs that determine the next state of the flip-flops in the memory. There may or may not be inputs to the combinational logic. There may also be output combinational logic, such as a decoder. If there is an input(s), it does not affect the outputs because they always correspond to and are dependent only on the present state of the memory. For the Mealy machine, the present state affects the outputs, just as in the Moore machine; but in addition, the inputs also affect the outputs. The outputs come directly from the combinational logic and not the memory.

Example of a Moore Machine

Figure 9–2(a) shows a Moore machine (modulus-26 binary counter with states 0 through 25) that is used to control the number of tablets (25) that go into each bottle in an assembly line. When the binary number in the memory (flip-flops) reaches binary twenty-five (11001), the counter recycles to 0 and the tablet flow and clock are cut off until the next bottle is in place. The combinational logic for the state transitions sets the modulus of the counter so that it sequences from binary state 0 to binary state 25, where 0 is the reset or rest state and the output combinational logic decodes binary state 25. There is no input in

this case, other than the clock, so the next state is determined only by the present state, which makes this a Moore machine. One tablet is bottled for each clock pulse. Once a bottle is in place, the first tablet is inserted at binary state 1, the second at binary state 2, and the twenty-fifth tablet when the binary state is 25. Count 25 is decoded and used to stop the flow of tablets and the clock. The counter stays in the 0 state until the next bottle is in position (indicated by a 1). Then the clock resumes, the count goes to 1, and the cycle repeats, as illustrated by the state diagram in Figure 9–2(b).

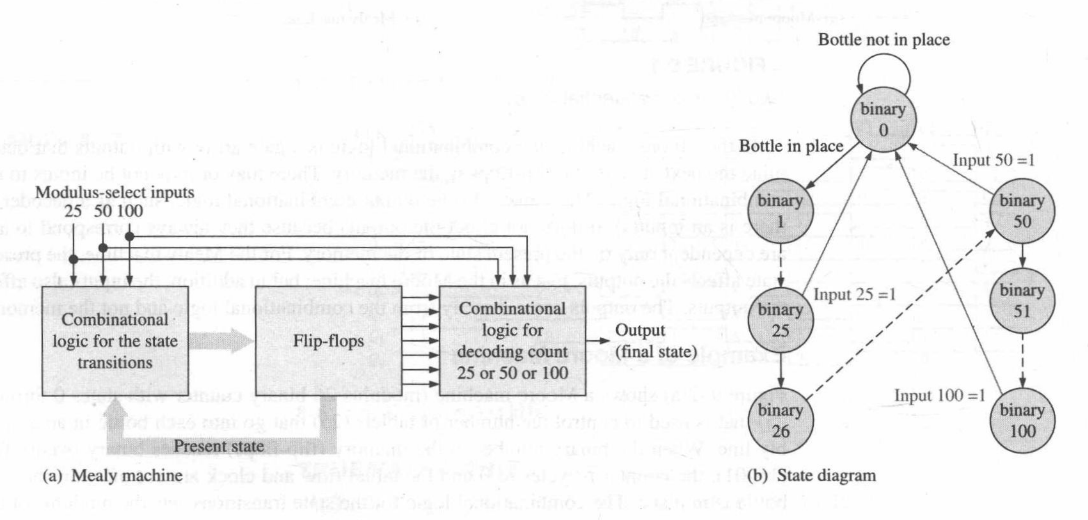


▲FIGURE 9-2

A fixed-modulus binary counter as an example of a Moore state machine. The dashed line in the state diagram means the states between binary 1 and 25 are not shown for simplicity.

Example of a Mealy Machine

Let's assume that the tablet-bottling system uses three different sizes of bottles: a 25-tablet bottle, a 50-tablet bottle, and a 100-tablet bottle. This operation requires a state machine with three different terminal counts: 25, 50, and 100. One approach is illustrated in Figure 9–3(a). The combinational logic sets the modulus of the counter depending on the modulus-select inputs. The output of the counter depends on both the present state and the modulus-select inputs, making this a Mealy machine. The state diagram is shown in part (b).



▲FIGURE 9-3

A variable-modulus binary counter as an example of a Mealy state machine. The red arrows in the state diagram represent the recycle paths that depend on the input number. The black dashed lines mean the interim states are not shown for simplicity.

SECTION 9-1 CHECKUP

Answers are at the end of the chapter.

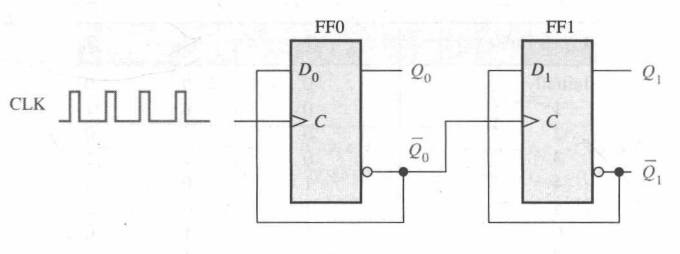
1. What characterizes a finite state machine?
2. Name the types of finite state machines.
3. Explain the difference between the two types of state machines.

9-2 Asynchronous Counters

A 2-Bit Asynchronous Binary Counter

The clock input of an asynchronous counter is always connected only to the LSB flip-flop.

Figure 9–4 shows a 2-bit counter connected for asynchronous operation. Notice that the clock (CLK) is applied to the clock input (C) of *only* the first flip-flop, FF0, which is always the least significant bit (LSB). The second flip-flop, FF1, is triggered by the $\overline{Q}_0$ output of FF0. FF0 changes state at the positive-going edge of each clock pulse, but FF1 changes only when triggered by a positive-going transition of the $\overline{Q}_0$ output of FF0. Because of the inherent propagation delay time through a flip-flop, a transition of the input clock pulse (CLK) and a transition of the $\overline{Q}_0$ output of FF0 can never occur at exactly the same time. Therefore, the two flip-flops are never simultaneously triggered, so the counter operation is asynchronous.



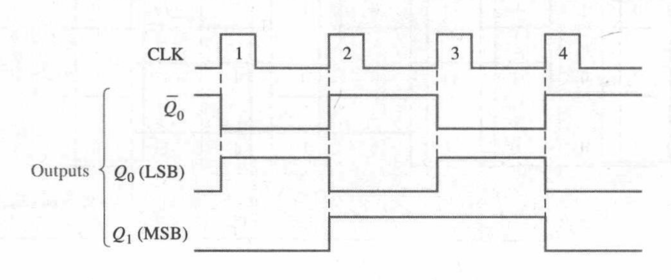
► **FIGURE 9-4**
A 2-bit asynchronous binary counter. Open file F09-04 to verify operation. *A Multisim tutorial is available on the website.*

The Timing Diagram

Let's examine the basic operation of the asynchronous counter of Figure 9–4 by applying four clock pulses to FF0 and observing the Q output of each flip-flop. Figure 9–5 illustrates the changes in the state of the flip-flop outputs in response to the clock pulses. Both flip-flops are connected for toggle operation ($D = \overline{Q}$) and are assumed to be initially RESET (Q LOW).

Asynchronous counters are also known as ripple counters.

The positive-going edge of CLK1 (clock pulse 1) causes the Q_0 output of FF0 to go HIGH, as shown in Figure 9–5. At the same time the $\overline{Q}_0$ output goes LOW, but it has no effect on FF1 because a positive-going transition must occur to trigger the flip-flop. After the leading edge of CLK1, $Q_0 = 1$ and $Q_1 = 0$. The positive-going edge of CLK2 causes Q_0 to go LOW. Output $\overline{Q}_0$ goes HIGH and triggers FF1, causing Q_1 to go HIGH. After the leading edge of CLK2, $Q_0 = 0$ and $Q_1 = 1$. The positive-going edge of CLK3 causes Q_0 to go HIGH again. Output $\overline{Q}_0$ goes LOW and has no effect on FF1. Thus, after the leading edge of CLK3, $Q_0 = 1$ and $Q_1 = 1$. The positive-going edge of CLK4 causes Q_0 to go LOW, while $\overline{Q}_0$ goes HIGH and triggers FF1, causing Q_1 to go LOW. After the leading edge of CLK4, $Q_0 = 0$ and $Q_1 = 0$. The counter has now recycled to its original state (both flip-flops are RESET).



► **FIGURE 9-5**
Timing diagram for the counter of Figure 9–4. As in previous chapters, output waveforms are shown in green.

In the timing diagram, the waveforms of the Q_0 and Q_1 outputs are shown relative to the clock pulses as illustrated in Figure 9–5. For simplicity, the transitions of Q_0, Q_1, and the clock pulses are shown as simultaneous even though this is an asynchronous counter. There is, of course, some small delay between the CLK and the Q_0 transition and between the $\overline{Q}_0$ transition and the Q_1 transition.

Note in Figure 9–5 that the 2-bit counter exhibits four different states, as you would expect with two flip-flops ($2^2 = 4$). Also, notice that if Q_0 represents the least significant bit (LSB) and Q_1 represents the most significant bit (MSB), the sequence of counter states represents a sequence of binary numbers as listed in Table 9–1.

In digital logic, Q_0 is always the LSB unless otherwise specified.

TABLE 9-1

Binary state sequence for the counter in Figure 9–4.

Clock Pulse	Q_1	Q_0
Initially	0	0
1	0	1
2	1	0
3	1	1
4 (recycles)	0	0

Since it goes through a binary sequence, the counter in Figure 9–4 is a binary counter. It actually counts the number of clock pulses up to three, and on the fourth pulse it recycles to its original state ($Q_0 = 0$, $Q_1 = 0$). The term **recycle** is commonly applied to counter operation; it refers to the transition of the counter from its final state back to its original state.

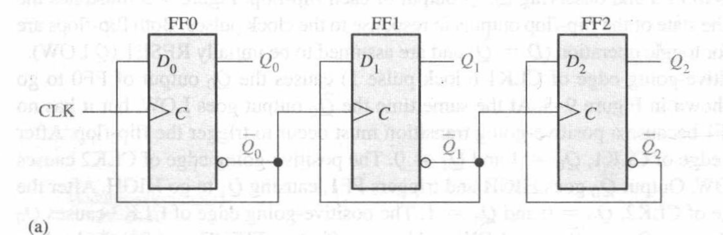
A 3-Bit Asynchronous Binary Counter

The state sequence for a 3-bit binary counter is listed in Table 9–2, and a 3-bit asynchronous binary counter is shown in Figure 9–6(a). The basic operation is the same as that of the 2-bit counter except that the 3-bit counter has eight states, due to its three flip-flops. A timing diagram is shown in Figure 9–6(b) for eight clock pulses. Notice that the counter progresses through a binary count of zero through seven and then recycles to the zero state. This counter can be easily expanded for higher count, by connecting additional toggle flip-flops.

TABLE 9-2

State sequence for a 3-bit binary counter.

Clock Pulse	Q_2	Q_1	Q_0
Initially	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8 (recycles)	0	0	0



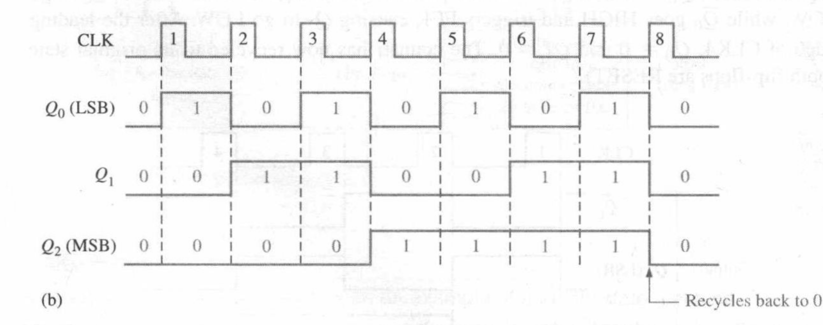


FIGURE 9-6

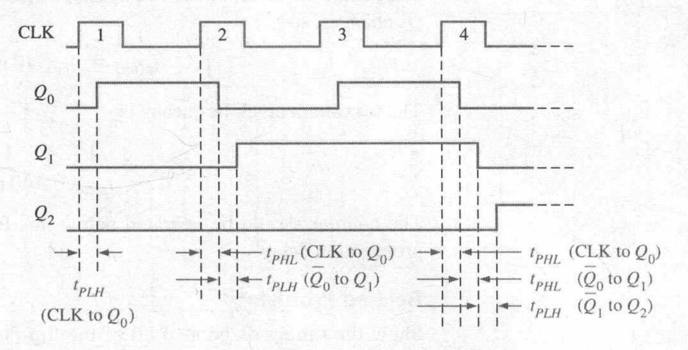
Three-bit asynchronous binary counter and its timing diagram for one cycle. Open file F09-06 to verify operation.

MultiSim

Propagation Delay

Asynchronous counters are commonly referred to as **ripple counters** for the following reason: The effect of the input clock pulse is first "felt" by FF0. This effect cannot get to FF1 immediately because of the propagation delay through FF0. Then there is the propagation delay through FF1 before FF2 can be triggered. Thus, the effect of an input clock pulse "ripples" through the counter, taking some time, due to propagation delays, to reach the last flip-flop.

To illustrate, notice that all three flip-flops in the counter of Figure 9–6 change state on the leading edge of CLK4. This ripple clocking effect is shown in Figure 9–7 for the first four clock pulses, with the propagation delays indicated. The LOW-to-HIGH transition of Q_0 occurs one delay time (t_{PLH}) after the positive-going transition of the clock pulse. The LOW-to-HIGH transition of Q_1 occurs one delay time (t_{PLH}) after the positive-going transition of $\overline{Q}_0$. The LOW-to-HIGH transition of Q_2 occurs one delay time (t_{PLH}) after the positive-going transition of $\overline{Q}_1$. As you can see, FF2 is not triggered until two delay times after the positive-going edge of the clock pulse, CLK4. Thus, it takes three propagation delay times for the effect of the clock pulse, CLK4, to ripple through the counter and change Q_2 from LOW to HIGH.



▲ **FIGURE 9-7**

Propagation delays in a 3-bit asynchronous (ripple-clocked) binary counter.

This cumulative delay of an asynchronous counter is a major disadvantage in many applications because it limits the rate at which the counter can be clocked and creates decoding problems. The maximum cumulative delay in a counter must be less than the period of the clock waveform.

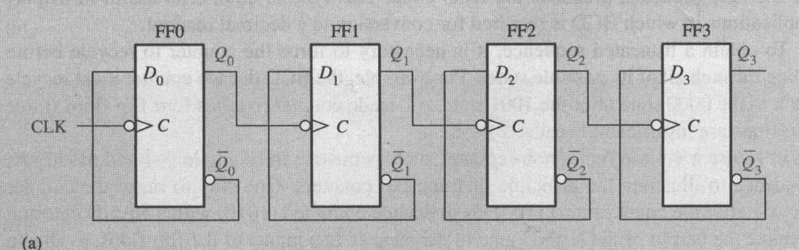
EXAMPLE 9-1

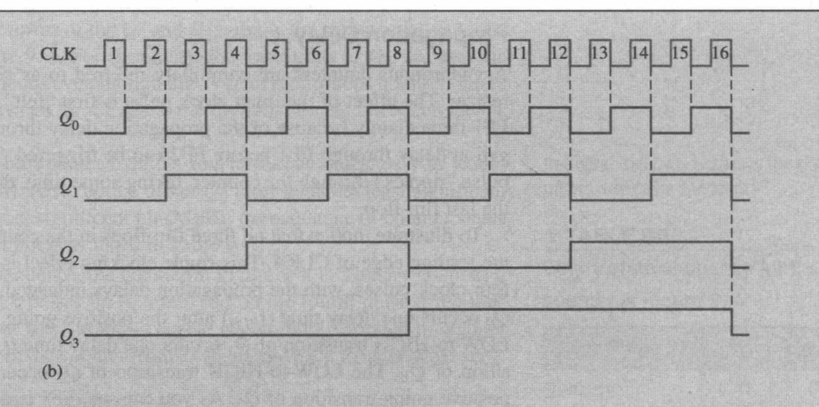
A 4-bit asynchronous binary counter is shown in Figure 9–8(a). Each D flip-flop is negative edge-triggered and has a propagation delay for 10 nanoseconds (ns). Develop a timing diagram showing the Q output of each flip-flop, and determine the total propagation delay time from the triggering edge of a clock pulse until a corresponding change can occur in the state of Q_3. Also determine the maximum clock frequency at which the counter can be operated.

► **FIGURE 9-8**

Four-bit asynchronous binary counter and its timing diagram. Open file F09-08 and verify the operation.

MultiSim





Solution

The timing diagram with delays omitted is as shown in Figure 9–8(b). For the total delay time, the effect of CLK8 or CLK16 must propagate through four flip-flops before Q_3 changes, so

$$t_{p(tot)} = 4 \times 10 \text{ ns} = \mathbf{40 \text{ ns}}$$

The maximum clock frequency is

$$f_{\max} = \frac{1}{t_{p(tot)}} = \frac{1}{40 \text{ ns}} = \mathbf{25 \text{ MHz}}$$

The counter should be operated below this frequency to avoid problems due to the propagation delay.

Related Problem*

Show the timing diagram if all of the flip-flops in Figure 9–8(a) are positive edge-triggered.

*Answers are at the end of the chapter.

Asynchronous Decade Counters

The **modulus** of a counter is the number of unique states through which the counter will sequence. The maximum possible number of states (maximum modulus) of a counter is 2^n, where n is the number of flip-flops in the counter. Counters can be designed to have a number of states in their sequence that is less than the maximum of 2^n. This type of sequence is called a *truncated sequence.*

A counter can have 2^n states, where n is the number of flip-flops.

One common modulus for counters with truncated sequences is ten (called MOD10). Counters with ten states in their sequence are called **decade** counters. A **decade counter** with a count sequence of zero (0000) through nine (1001) is a BCD decade counter because its ten-state sequence produces the BCD code. This type of counter is useful in display applications in which BCD is required for conversion to a decimal readout.

To obtain a truncated sequence, it is necessary to force the counter to recycle before going through all of its possible states. For example, the BCD decade counter must recycle back to the 0000 state after the 1001 state. A decade counter requires four flip-flops (three flip-flops are insufficient because $2^3 = 8$).

Let's use a 4-bit asynchronous counter such as the one in Example 9–1 and modify its sequence to illustrate the principle of truncated counters. One way to make the counter recycle after the count of nine (1001) is to decode count ten (1010) with a NAND gate and connect the output of the NAND gate to the clear ($\overline{CLR}$) inputs of the flip-flops, as shown in Figure 9–9(a).

Partial Decoding

Notice in Figure 9–9(a) that only Q_1 and Q_3 are connected to the NAND gate inputs. This arrangement is an example of *partial decoding,* in which the two unique states ($Q_1 = 1$ and $Q_3 = 1$) are sufficient to decode the count of ten because none of the other states (zero through nine) have both Q_1 and Q_3 HIGH at the same time. When the counter goes into count ten (1010), the decoding gate output goes LOW and asynchronously resets all the flip-flops.

The resulting timing diagram is shown in Figure 9–9(b). Notice that there is a glitch on the Q_1 waveform. The reason for this glitch is that Q_1 must first go HIGH before the count of ten can be decoded. Not until several nanoseconds after the counter goes to the count of ten does the output of the decoding gate go LOW (both inputs are HIGH). Thus, the counter is in the 1010 state for a short time before it is reset to 0000, thus producing the glitch on Q_1 and the resulting glitch on the $\overline{CLR}$ line that resets the counter.

Other truncated sequences can be implemented in a similar way, as Example 9–2 shows.

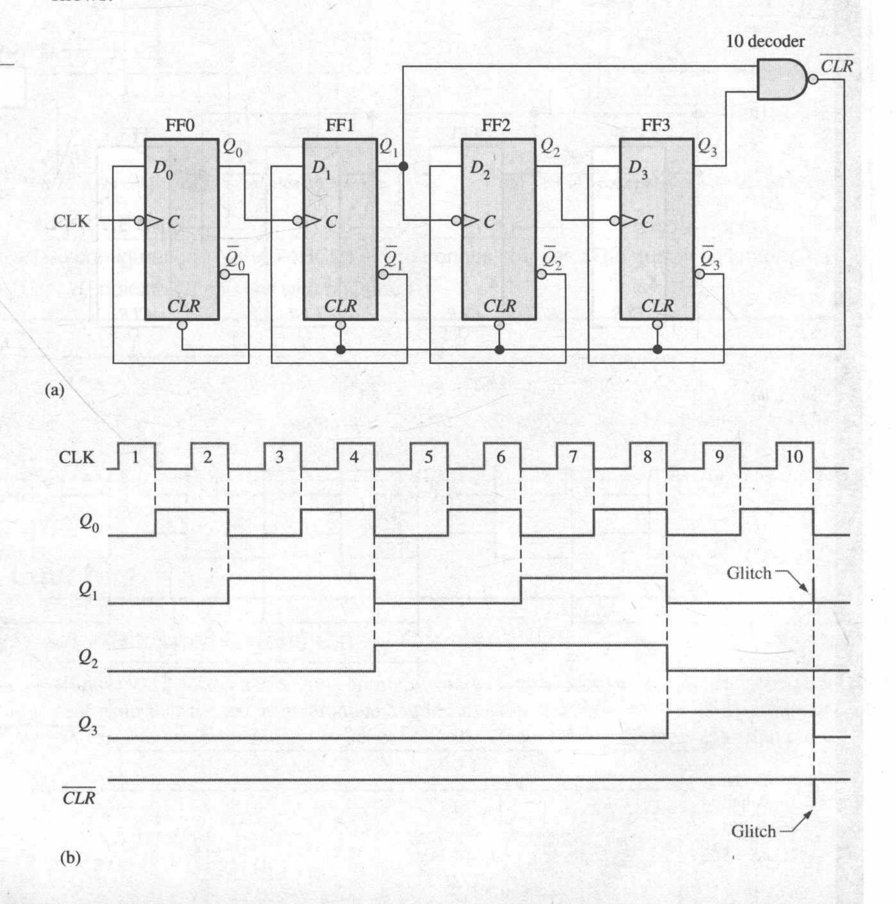


FIGURE 9-9 An asynchronously clocked decade counter with asynchronous recycling.

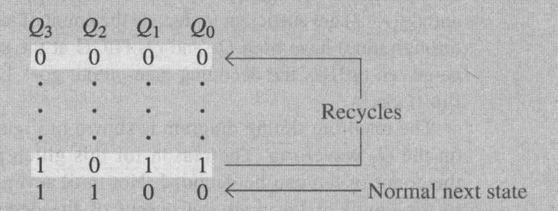
EXAMPLE 9-2

Show how an asynchronous counter with J-K flip-flops can be implemented having a modulus of twelve with a straight binary sequence from 0000 through 1011.

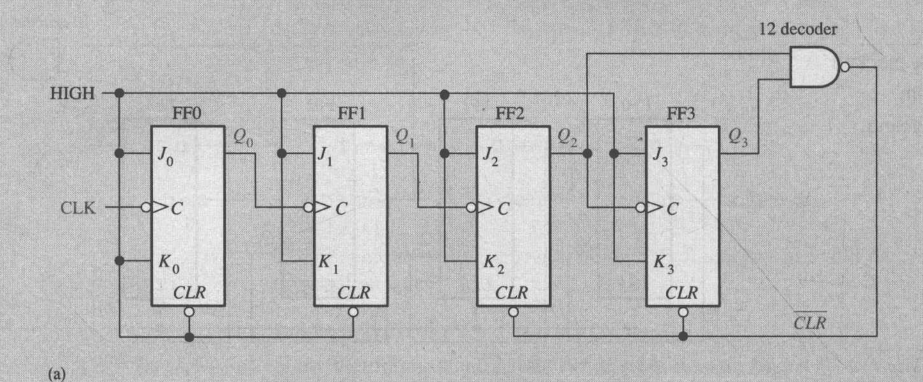
Solution

Since three flip-flops can produce a maximum of eight states, four flip-flops are required to produce any modulus greater than eight but less than or equal to sixteen.

When the counter gets to its last state, 1011, it must recycle back to 0000 rather than going to its normal next state of 1100, as illustrated in the following sequence chart:



Observe that Q_0 and Q_1 both go to 0 anyway, but Q_2 and Q_3 must be forced to 0 on the twelfth clock pulse. Figure 9–10(a) shows the modulus-12 counter. The NAND gate partially decodes count twelve (1100) and resets flip-flop 2 and flip-flop 3. Thus, on the twelfth clock pulse, the counter is forced to recycle from count eleven to count zero, as shown in the timing diagram of Figure 9–10(b). (It is in count twelve for only a few nanoseconds before it is reset by the glitch on $\overline{CLR}$.)



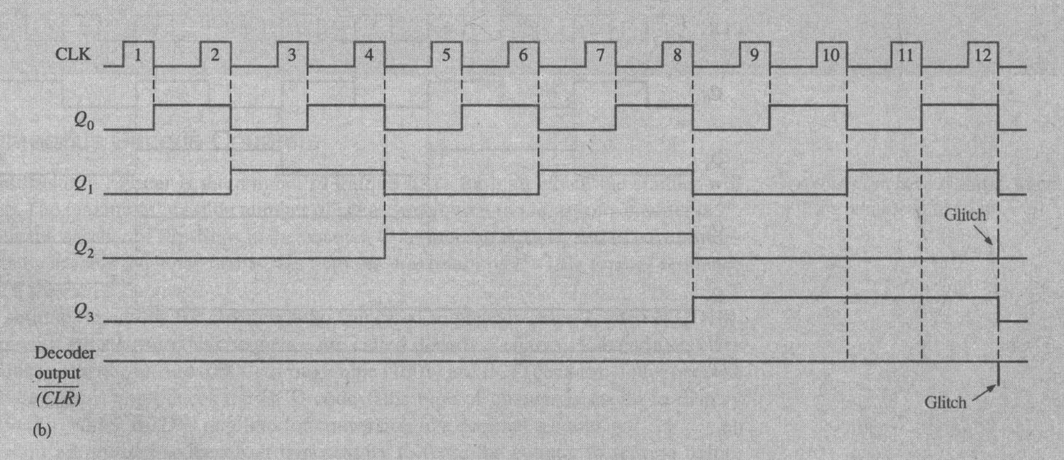


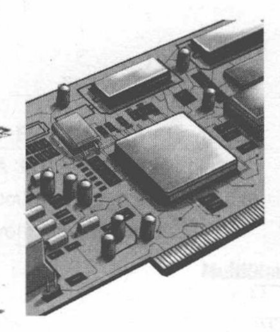
FIGURE 9-10

Asynchronously clocked modulus-12 counter with asynchronous recycling.

Related Problem

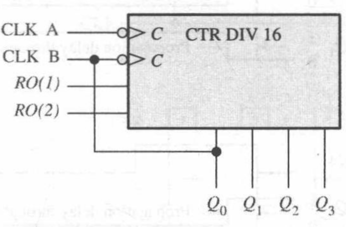
How can the counter in Figure 9–10(a) be modified to make it a modulus-13 counter?

IMPLEMENTATION: 4-BIT ASYNCHRONOUS BINARY COUNTER

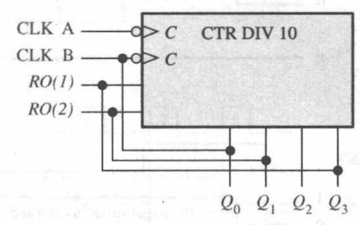


Fixed-Function Device The 74HC93 is an example of a specific integrated circuit asynchronous counter. This device actually consists of a single flip-flop (CLK A) and a 3-bit asynchronous counter (CLK B). This arrangement is for flexibility. It can be used as a divide-by-2 device if only the single flip-flop is used, or it can be used as a modulus-8 counter if only the 3-bit counter portion is used. This device also provides gated reset inputs, *RO(1)* and *RO(2)*. When both of these inputs are HIGH, the counter is reset to the 0000 state $\overline{CLR}$.

Additionally, the 74HC93 can be used as a 4-bit modulus-16 counter (counts 0 through 15) by connecting the Q_0 output to the CLK B input as shown by the logic symbol in Figure 9–11(a). It can also be configured as a decade counter (counts 0 through 9) with asynchronous recycling by using the gated reset inputs for partial decoding of count ten, as shown by the logic symbol in Figure 9–11(b).



(a) 74HC93 connected as a modulus-16 counter



(b) 74HC93 connected as a decade counter

▲ **FIGURE 9-11**

Two configurations of the 74HC93 asynchronous counter. (The qualifying label, CTR DIV *n*, indicates a counter with *n* states.)

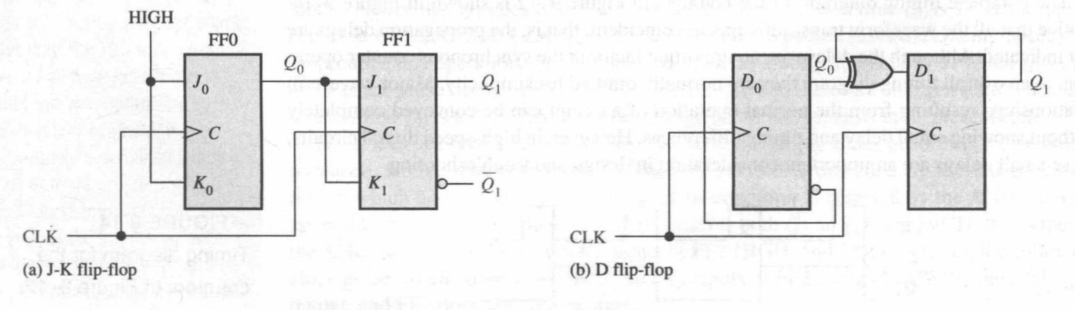
SECTION 9-2 CHECKUP

1. What does the term *asynchronous* mean in relation to counters?
2. How many states does a modulus-14 counter have? What is the minimum number of flip-flops required?

9-3 Synchronous Counters

A 2-Bit Synchronous Binary Counter

Figure 9–12 shows a 2-bit synchronous binary counter. Notice that an arrangement different from that for the asynchronous counter must be used for the J_1 and K_1 inputs of FF1 in order to achieve a binary sequence. A D flip-flop implementation is shown in part (b).



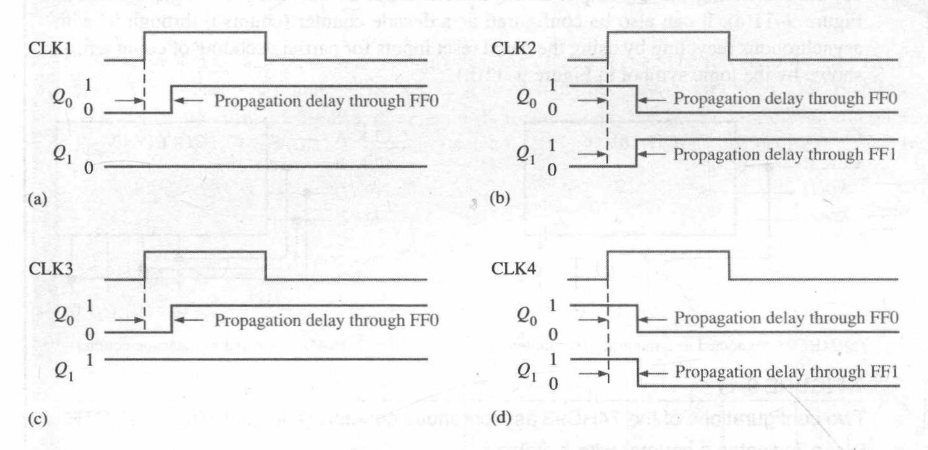
(a) J-K flip-flop (b) D flip-flop

▲ **FIGURE 9-12**

2-bit synchronous binary counters.

The clock input goes to each flip-flop in a synchronous counter.

The operation of a J-K flip-flop synchronous counter is as follows: First, assume that the counter is initially in the binary 0 state; that is, both flip-flops are RESET. When the positive edge of the first clock pulse is applied, FF0 will toggle and Q_0 will therefore go HIGH. What happens to FF1 at the positive-going edge of CLK1? To find out, let's look at the input conditions of FF1. Inputs J_1 and K_1 are both LOW because Q_0, to which they are connected, has not yet gone HIGH. Remember, there is a propagation delay from the triggering edge of the clock pulse until the Q output actually makes a transition. So, $J = 0$ and $K = 0$ when the leading edge of the first clock pulse is applied. This is a no-change condition, and therefore FF1 does not change state. A timing detail of this portion of the counter operation is shown in Figure 9–13(a).



▲ **FIGURE 9-13**

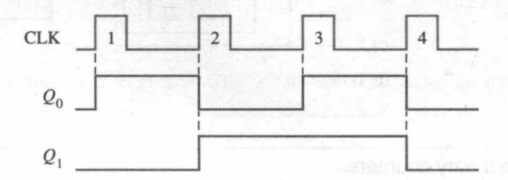
Timing details for the 2-bit synchronous counter operation (the propagation delays of both flip-flops are assumed to be equal).

After CLK1, $Q_0 = 1$ and $Q_1 = 0$ (which is the binary 1 state). When the leading edge of CLK2 occurs, FF0 will toggle and Q_0 will go LOW. Since FF1 has a HIGH ($Q_0 = 1$) on its J_1 and K_1 inputs at the triggering edge of this clock pulse, the flip-flop toggles and Q_1 goes HIGH. Thus, after CLK2, $Q_0 = 0$ and $Q_1 = 1$ (which is a binary 2 state). The timing detail for this condition is shown in Figure 9–13(b).

When the leading edge of CLK3 occurs, FF0 again toggles to the SET state ($Q_0 = 1$), and FF1 remains SET ($Q_1 = 1$) because its J_1 and K_1 inputs are both LOW ($Q_0 = 0$). After this triggering edge, $Q_0 = 1$ and $Q_1 = 1$ (which is a binary 3 state). The timing detail is shown in Figure 9–13(c).

Finally, at the leading edge of CLK4, Q_0 and Q_1 go LOW because they both have a toggle condition on their J and K inputs. The timing detail is shown in Figure 9–13(d). The counter has now recycled to its original state, binary 0. Examination of the D flip-flop counter in Figure 9–12(b) will show the timing diagram is the same as for the J-K flip-flop counter.

The complete timing diagram for the counters in Figure 9–12 is shown in Figure 9–14. Notice that all the waveform transitions appear coincident; that is, the propagation delays are not indicated. Although the delays are an important factor in the synchronous counter operation, in an overall timing diagram they are normally omitted for simplicity. Major waveform relationships resulting from the normal operation of a circuit can be conveyed completely without showing small delay and timing differences. However, in high-speed digital circuits, these small delays are an important consideration in design and troubleshooting.



◀ **FIGURE 9-14**

Timing diagram for the counters of Figure 9–12.

A 3-Bit Synchronous Binary Counter

A 3-bit synchronous binary counter is shown in Figure 9–15, and its timing diagram is shown in Figure 9–16. You can understand this counter operation by examining its sequence of states as shown in Table 9–3.

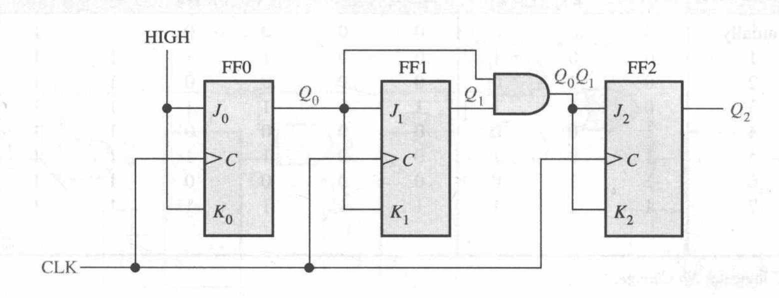


FIGURE 9-15
A 3-bit synchronous binary counter. Open file F09-15 to verify the operation.

MultiSim

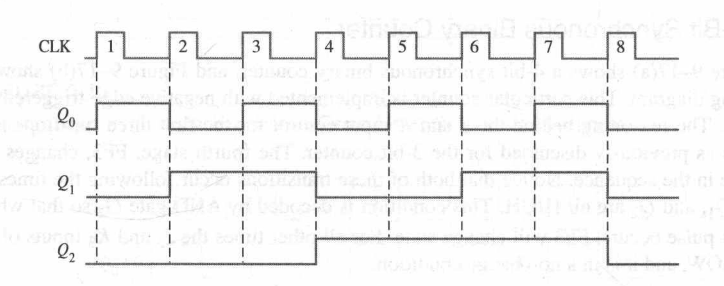


FIGURE 9-16
Timing diagram for the counter of Figure 9–15.

TABLE 9-3
State sequence for a 3-bit binary counter.

Clock Pulse	Q_2	Q_1	Q_0
Initially	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8 (recycles)	0	0	0

InfoNote

The TSC or *time stamp counter* in some microprocessors is used for performance monitoring, which enables a number of parameters important to the overall performance of a system to be determined exactly. By reading the TSC before and after the execution of a procedure, the precise time required for the procedure can be determined based on the processor cycle time. In this way, the TSC forms the basis for all time evaluations in connection with optimizing system operation. For example, it can be accurately determined which of two or more programming sequences is more efficient. This is a very useful tool for compiler developers and system programmers in producing the most effective code.

First, let's look at Q_0. Notice that Q_0 changes on each clock pulse as the counter progresses from its original state to its final state and then back to its original state. To produce this operation, FF0 must be held in the toggle mode by constant HIGHs on its J_0 and K_0 inputs. Notice that Q_1 goes to the opposite state following each time Q_0 is a 1. This change occurs at CLK2, CLK4, CLK6, and CLK8. The CLK8 pulse causes the counter to recycle. To produce this operation, Q_0 is connected to the J_1 and K_1 inputs of FF1. When Q_0 is a 1 and a clock pulse occurs, FF1 is in the toggle mode and therefore changes state. The other times, when Q_0 is a 0, FF1 is in the no-change mode and remains in its present state.

Next, let's see how FF2 is made to change at the proper times according to the binary sequence. Notice that both times Q_2 changes state, it is preceded by the unique condition in which both Q_0 and Q_1 are HIGH. This condition is detected by the AND gate and applied to the J_2 and K_2 inputs of FF2. Whenever both Q_0 and Q_1 are HIGH, the output of the AND gate makes the J_2 and K_2 inputs of FF2 HIGH, and FF2 toggles on the following clock pulse. At all other times, the J_2 and K_2 inputs of FF2 are held LOW by the AND gate output, and FF2 does not change state.

The analysis of the counter in Figure 9–15 is summarized in Table 9–4.

▼TABLE 9-4

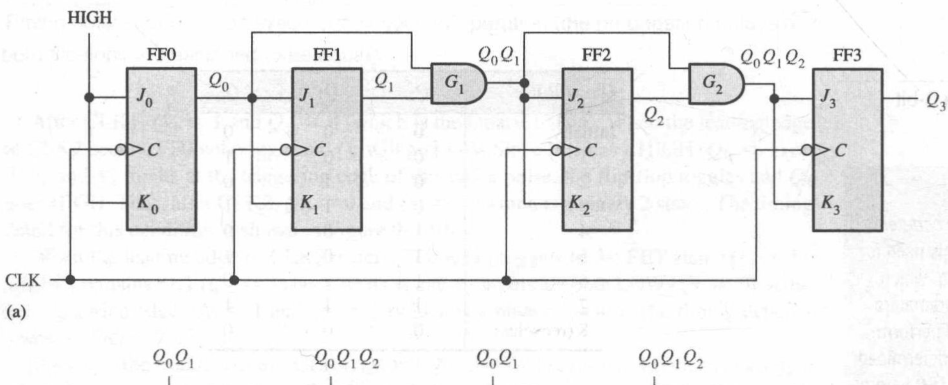
Summary of the analysis of the counter in Figure 9–15.

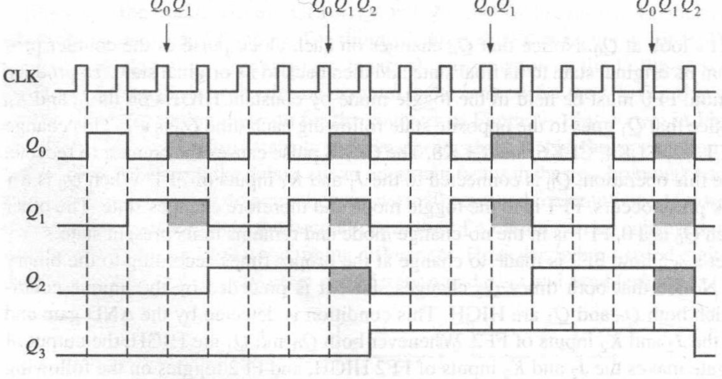
Clock Pulse	Outputs Q_2	Q_1	Q_0	*J-K* Inputs J_2	K_2	J_1	K_1	J_0	K_0	At the Next Clock Pulse FF2	FF1	FF0
Initially	0	0	0	0	0	0	0	1	1	NC*	NC	Toggle
1	0	0	1	0	0	1	1	1	1	NC	Toggle	Toggle
2	0	1	0	0	0	0	0	1	1	NC	NC	Toggle
3	0	1	1	1	1	1	1	1	1	Toggle	Toggle	Toggle
4	1	0	0	0	0	0	0	1	1	NC	NC	Toggle
5	1	0	1	0	0	1	1	1	1	NC	Toggle	Toggle
6	1	1	0	0	0	0	0	1	1	NC	NC	Toggle
7	1	1	1	1	1	1	1	1	1	Toggle	Toggle	Toggle
										Counter recycles back to 000.		

*NC indicates *No Change*.

A 4-Bit Synchronous Binary Counter

Figure 9–17(a) shows a 4-bit synchronous binary counter, and Figure 9–17(b) shows its timing diagram. This particular counter is implemented with negative edge-triggered flip-flops. The reasoning behind the J and K input control for the first three flip-flops is the same as previously discussed for the 3-bit counter. The fourth stage, FF3, changes only twice in the sequence. Notice that both of these transitions occur following the times that Q_0, Q_1, and Q_2 are all HIGH. This condition is decoded by AND gate G_2 so that when a clock pulse occurs, FF3 will change state. For all other times the J_3 and K_3 inputs of FF3 are LOW, and it is in a no-change condition.





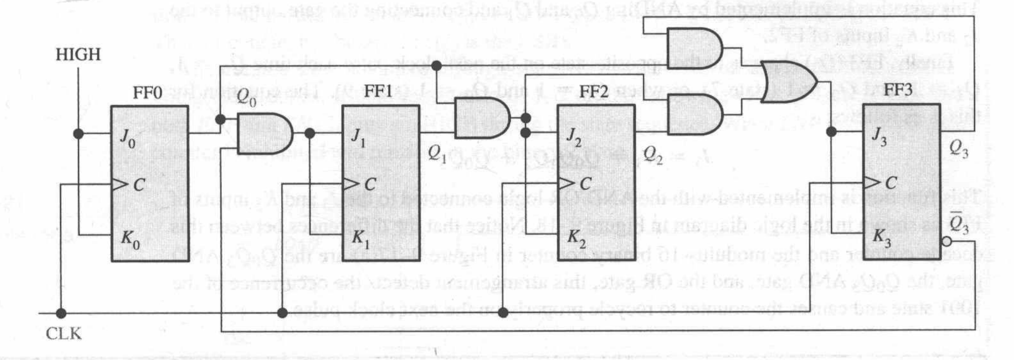
▲FIGURE 9-17

A 4-bit synchronous binary counter and timing diagram. Times where the AND gate outputs are HIGH are indicated by the shaded areas.

A 4-Bit Synchronous Decade Counter

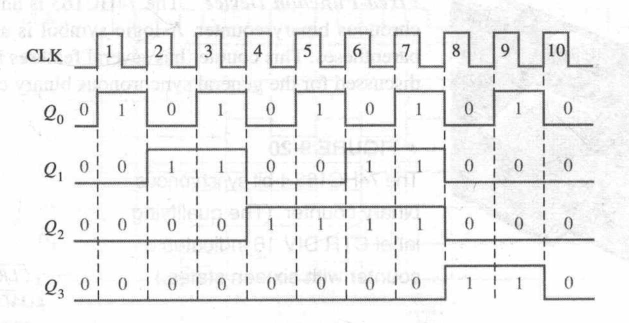
A decade counter has ten states.

As you know, a BCD decade counter exhibits a truncated binary sequence and goes from 0000 through the 1001 state. Rather than going from the 1001 state to the 1010 state, it recycles to the 0000 state. A synchronous BCD decade counter is shown in Figure 9–18. The timing diagram for the decade counter is shown in Figure 9–19.



MultiSim

▲ **FIGURE 9-18**
A synchronous BCD decade counter. Open file F09-18 to verify operation.



▶ **FIGURE 9-19**
Timing diagram for the BCD decade counter (Q_0 is the LSB).

The counter operation is shown by the sequence of states in Table 9–5. First, notice that FF0 (Q_0) toggles on each clock pulse, so the logic equation for its J_0 and K_0 inputs is

$$J_0 = K_0 = 1$$

This equation is implemented by connecting J_0 and K_0 to a constant HIGH level.

▶ **TABLE 9-5**
States of a BCD decade counter.

Clock Pulse	Q_3	Q_2	Q_1	Q_0
Initially	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10 (recycles)	0	0	0	0

Next, notice in Table 9–5 that FF1 (Q_1) changes on the next clock pulse each time $Q_0 = 1$ and $Q_3 = 0$, so the logic equation for the J_1 and K_1 inputs is

$$J_1 = K_1 = Q_0\overline{Q}_3$$

This equation is implemented by ANDing Q_0 and $\overline{Q}_3$ and connecting the gate output to the J_1 and K_1 inputs of FF1.

Flip-flop 2 (Q_2) changes on the next clock pulse each time both $Q_0 = 1$ and $Q_1 = 1$. This requires an input logic equation as follows:

$$J_2 = K_2 = Q_0Q_1$$

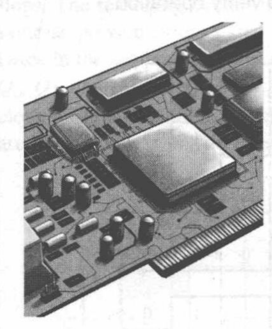
This equation is implemented by ANDing Q_0 and Q_1 and connecting the gate output to the J_2 and K_2 inputs of FF2.

Finally, FF3 (Q_3) changes to the opposite state on the next clock pulse each time $Q_0 = 1$, $Q_1 = 1$, and $Q_2 = 1$ (state 7), or when $Q_0 = 1$ and $Q_3 = 1$ (state 9). The equation for this is as follows:

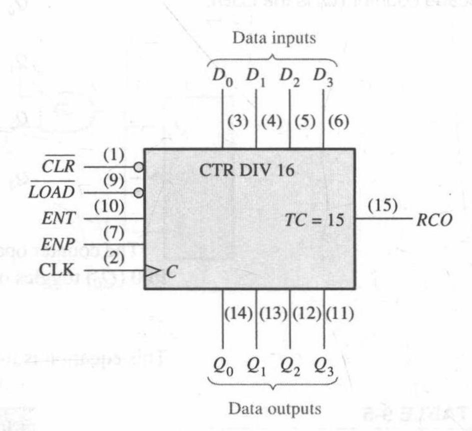
$$J_3 = K_3 = Q_0Q_1Q_2 + Q_0Q_3$$

This function is implemented with the AND/OR logic connected to the J_3 and K_3 inputs of FF3 as shown in the logic diagram in Figure 9–18. Notice that the differences between this decade counter and the modulus-16 binary counter in Figure 9–17(a) are the $Q_0\overline{Q}_3$ AND gate, the Q_0Q_3 AND gate, and the OR gate; this arrangement detects the occurrence of the 1001 state and causes the counter to recycle properly on the next clock pulse.

IMPLEMENTATION: 4-BIT SYNCHRONOUS BINARY COUNTER



Fixed-Function Device The 74HC163 is an example of an integrated circuit 4-bit synchronous binary counter. A logic symbol is shown in Figure 9–20 with pin numbers in parentheses. This counter has several features in addition to the basic functions previously discussed for the general synchronous binary counter.



▶ FIGURE 9-20
The 74HC163 4-bit synchronous binary counter. (The qualifying label CTR DIV 16 indicates a counter with sixteen states.)

First, the counter can be synchronously preset to any 4-bit binary number by applying the proper levels to the parallel data inputs. When a LOW is applied to the $\overline{LOAD}$ input, the counter will assume the state of the data inputs on the next clock pulse. Thus, the counter sequence can be started with any 4-bit binary number.

Also, there is an active-LOW clear input ($\overline{CLR}$), which synchronously resets all four flip-flops in the counter. There are two enable inputs, *ENP* and *ENT*. These inputs must both be HIGH for the counter to sequence through its binary states. When at least one input is LOW, the counter is disabled. The ripple clock output (*RCO*) goes HIGH when the counter reaches the last state in its sequence of fifteen, called the **terminal count** ($TC = 15$). This output, in conjunction with the enable inputs, allows these counters to be cascaded for higher count sequences.

Figure 9–21 shows a timing diagram of this counter being preset to twelve (1100) and then counting up to its terminal count, fifteen (1111). Input D_0 is the least significant input bit, and Q_0 is the least significant output bit.

Let's examine this timing diagram in detail. This will aid you in interpreting timing diagrams in this chapter or on manufacturers' data sheets. To begin, the LOW level pulse on the $\overline{CLR}$ input causes all the outputs (Q_0, Q_1, Q_2, and Q_3) to go LOW.

Next, the LOW level pulse on the $\overline{LOAD}$ input synchronously enters the data on the data inputs (D_0, D_1, D_2, and D_3) into the counter. These data appear on the Q outputs at the time of the first positive-going clock edge after $\overline{LOAD}$ goes LOW. This is the preset operation. In this particular example, Q_0 is LOW, Q_1 is LOW, Q_2 is HIGH, and Q_3 is HIGH. This, of course, is a binary 12 (Q_0 is the LSB).

The counter now advances through states 13, 14, and 15 on the next three positive-going clock edges. It then recycles to 0, 1, 2 on the following clock pulses. Notice that both *ENP* and *ENT* inputs are HIGH during the state sequence. When *ENP* goes LOW, the counter is inhibited and remains in the binary 2 state.

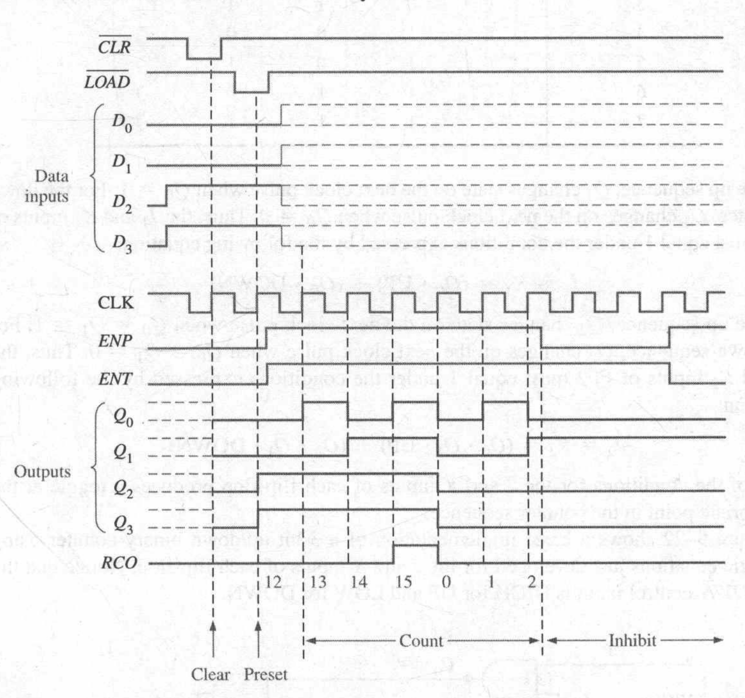


FIGURE 9-21
Timing example for a 74HC163.

SECTION 9-3 CHECKUP

1. How does a synchronous counter differ from an asynchronous counter?
2. Explain the function of the preset feature of counters such as the 74HC163.
3. Describe the purpose of the *ENP* and *ENT* inputs and the *RCO* output for the 74HC163 counter.

9-4 Up/Down Synchronous Counters

In general, most up/down counters can be reversed at any point in their sequence. For instance, the 3-bit binary counter can be made to go through the following sequence:

$$\overbrace{0, 1, 2, 3, 4, 5,}^{\text{UP}}\underbrace{4, 3, 2,}_{\text{DOWN}}\overbrace{3, 4, 5, 6, 7,}^{\text{UP}}\underbrace{6, 5,}_{\text{DOWN}}\text{ etc.}$$

Table 9–6 shows the complete up/down sequence for a 3-bit binary counter. The arrows indicate the state-to-state movement of the counter for both its UP and its DOWN modes of operation. An examination of Q_0 for both the up and down sequences shows that FF0 toggles on each clock pulse. Thus, the J_0 and K_0 inputs of FF0 are

$$J_0 = K_0 = 1$$

Clock Pulse	Up	Q_2	Q_1	Q_0	Down
0		0	0	0	
1		0	0	1	
2		0	1	0	
3		0	1	1	
4		1	0	0	
5		1	0	1	
6		1	1	0	
7		1	1	1	

◄ **TABLE 9-6**

Up/Down sequence for a 3-bit binary counter.

For the up sequence, Q_1 changes state on the next clock pulse when $Q_0 = 1$. For the down sequence, Q_1 changes on the next clock pulse when $Q_0 = 0$. Thus, the J_1 and K_1 inputs of FF1 must equal 1 under the conditions expressed by the following equation:

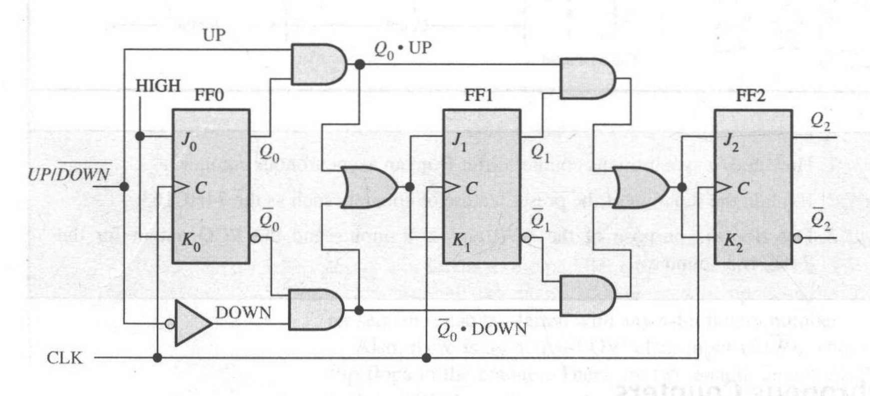
$$J_1 = K_1 = (Q_0 \cdot \text{UP}) + (\overline{Q}_0 \cdot \text{DOWN})$$

For the up sequence, Q_2 changes state on the next clock pulse when $Q_0 = Q_1 = 1$. For the down sequence, Q_2 changes on the next clock pulse when $Q_0 = Q_1 = 0$. Thus, the J_2 and K_2 inputs of FF2 must equal 1 under the conditions expressed by the following equation:

$$J_2 = K_2 = (Q_0 \cdot Q_1 \cdot \text{UP}) + (\overline{Q}_0 \cdot \overline{Q}_1 \cdot \text{DOWN})$$

Each of the conditions for the J and K inputs of each flip-flop produces a toggle at the appropriate point in the counter sequence.

Figure 9–22 shows a basic implementation of a 3-bit up/down binary counter using the logic equations just developed for the J and K inputs of each flip-flop. Notice that the $UP/\overline{DOWN}$ control input is HIGH for UP and LOW for DOWN.



▲ **FIGURE 9-22**

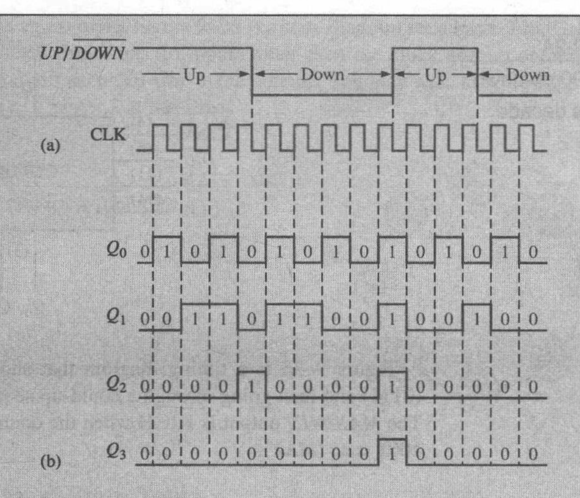
A basic 3-bit up/down synchronous counter. Open file F09-22 to verify operation.

MultiSim

EXAMPLE 9-3

Show the timing diagram and determine the sequence of a 4-bit synchronous binary up/down counter if the clock and $UP/\overline{DOWN}$ control inputs have waveforms as shown in Figure 9–23(a). The counter starts in the all-0s state and is positive edge-triggered.

FIGURE 9-23



Solution

The timing diagram showing the Q outputs is shown in Figure 9–23(b). From these waveforms, the counter sequence is as shown in Table 9–7.

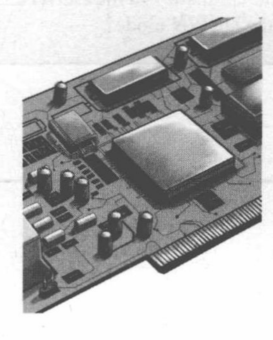
TABLE 9-7

Q_3	Q_2	Q_1	Q_0	
0	0	0	0	
0	0	0	1	
0	0	1	0	UP
0	0	1	1	
0	1	0	0	
0	0	1	1	
0	0	1	0	
0	0	0	1	DOWN
0	0	0	0	
1	1	1	1	
0	0	0	0	
0	0	0	1	UP
0	0	1	0	
0	0	0	1	DOWN
0	0	0	0	

Related Problem

Show the timing diagram if the $UP/\overline{DOWN}$ control waveform in Figure 9–23(a) is inverted.

IMPLEMENTATION: UP/DOWN DECADE COUNTER



Fixed-Function Device Figure 9–24 shows a logic diagram for the 74HC190, an example of an integrated circuit up/down synchronous decade counter. The direction of the count is determined by the level of the up/down input ($D/\overline{U}$). When this input is HIGH, the counter counts down; when it is LOW, the counter counts up. Also, this device can be preset to any desired BCD digit as determined by the states of the data inputs when the $\overline{LOAD}$ input is LOW.

The *MAX/MIN* output produces a HIGH pulse when the terminal count nine (1001) is reached in the UP mode or when the terminal count zero (0000) is reached in the DOWN mode. The *MAX/MIN* output, the ripple clock output ($\overline{RCO}$), and the count enable input ($\overline{CTEN}$) are used when cascading counters. (Cascaded counters are discussed in Section 9–6.)

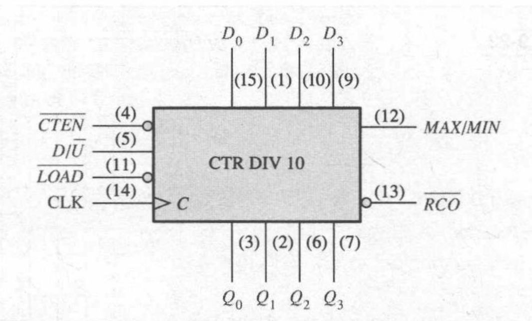


FIGURE 9-24 The 74HC190 up/down synchronous decade counter.

Figure 9–25 is a timing diagram that shows the 74HC190 counter preset to seven (0111) and then going through a count-up sequence followed by a count-down sequence. The *MAX/MIN* output is HIGH when the counter is in either the all-0s state (*MIN*) or the 1001 state (*MAX*).

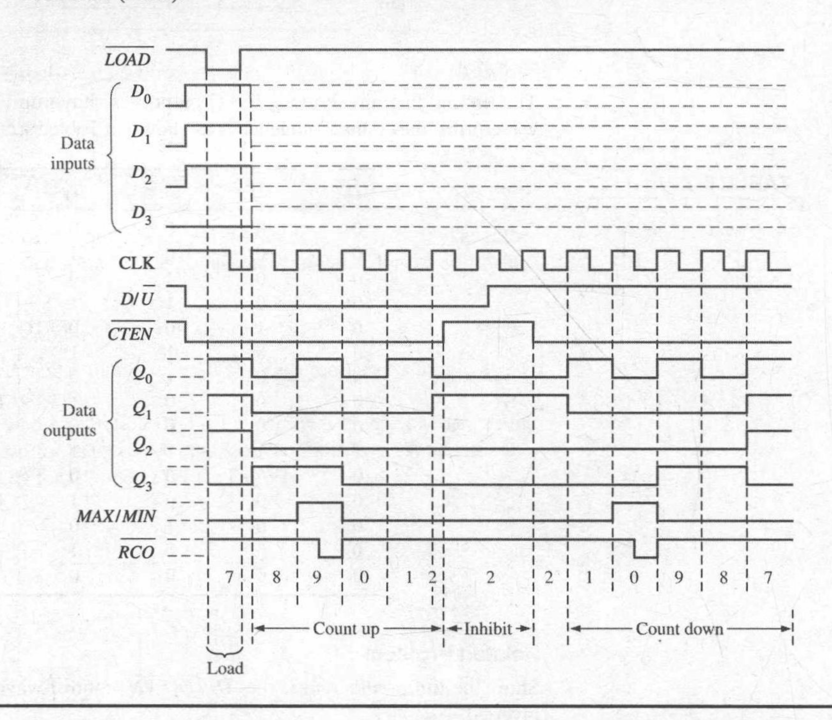


FIGURE 9-25 Timing example for a 74HC190.

SECTION 9-4 CHECKUP

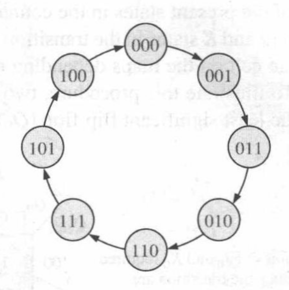
1. A 4-bit up/down binary counter is in the DOWN mode and in the 1010 state. On the next clock pulse, to what state does the counter go?
2. What is the terminal count of a 4-bit binary counter in the UP mode? In the DOWN mode? What is the next state after the terminal count in the DOWN mode?

9-5 Design of Synchronous Counters

Step 1: State Diagram

The first step in the design of a state machine (counter) is to create a state diagram. A **state diagram** shows the progression of states through which the counter advances when it is

clocked. As an example, Figure 9–26 is a state diagram for a basic 3-bit Gray code counter. This particular circuit has no inputs other than the clock and no outputs other than the outputs taken off each flip-flop in the counter. You may wish to review the coverage of the Gray code in Chapter 2 at this time.



▶ FIGURE 9-26
State diagram for a 3-bit Gray code counter.

Step 2: Next-State Table

Once the sequential circuit is defined by a state diagram, the second step is to derive a next-state table, which lists each state of the counter (present state) along with the corresponding next state. *The next state is the state that the counter goes to from its present state upon application of a clock pulse.* The next-state table is derived from the state diagram and is shown in Table 9–8 for the 3-bit Gray code counter. Q_0 is the least significant bit.

▼ TABLE 9-8
Next-state table for 3-bit Gray code counter.

Present State			Next State		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	0	1
0	0	1	0	1	1
0	1	1	0	1	0
0	1	0	1	1	0
1	1	0	1	1	1
1	1	1	1	0	1
1	0	1	1	0	0
1	0	0	0	0	0

▼ TABLE 9-9
Transition table for a J-K flip-flop.

Output Transitions			Flip-Flop Inputs	
Q_N		Q_{N+1}	J	K
0	⟶	0	0	X
0	⟶	1	1	X
1	⟶	0	X	1
1	⟶	1	X	0

Q_N: present state
Q_{N+1}: next state
X: "don't care"

Step 3: Flip-Flop Transition Table

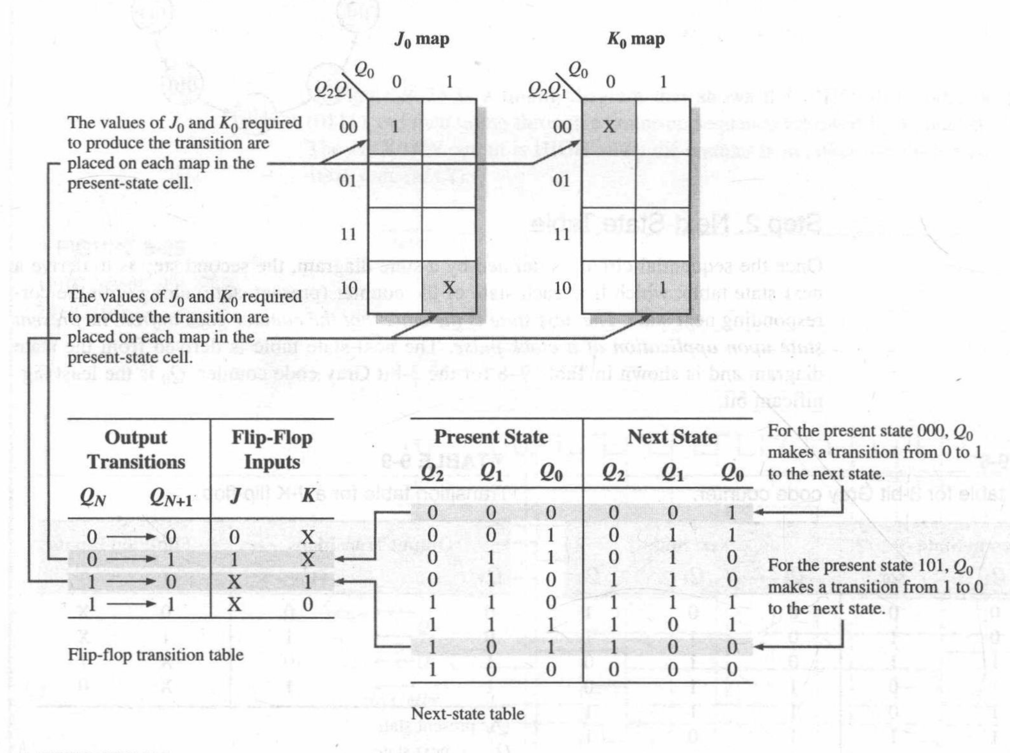
Table 9–9 is a transition table for the J-K flip-flop. All possible output transitions are listed by showing the Q output of the flip-flop going from present states to next states. Q_N is the present state of the flip-flop (before a clock pulse) and Q_{N+1} is the next state (after a clock pulse). For each output transition, the J and K inputs that will cause the transition to occur are listed. An X indicates a "don't care" (the input can be either a 1 or a 0).

To design the counter, the transition table is applied to each of the flip-flops in the counter, based on the next-state table (Table 9–8). For example, for the present state 000, Q_0 goes from a present state of 0 to a next state of 1. To make this happen, J_0 must be a 1 and you don't care what K_0 is ($J_0 = 1$, $K_0 = X$), as you can see in the transition table (Table 9–9). Next, Q_1 is 0 in the present state and remains a 0 in the next state. For this transition, $J_1 = 0$ and $K_1 = X$. Finally, Q_2 is 0 in the present state and remains a 0 in the next state. Therefore, $J_2 = 0$ and $K_2 = X$. This analysis is repeated for each present state in Table 9–8.

Step 4: Karnaugh Maps

Karnaugh maps can be used to determine the logic required for the J and K inputs of each flip-flop in the counter. There is a Karnaugh map for the J input and a Karnaugh map for the K input of each flip-flop. In this design procedure, each cell in a Karnaugh map represents one of the present states in the counter sequence listed in Table 9–8.

From the J and K states in the transition table (Table 9–9) a 1, 0, or X is entered into each present-state cell on the maps depending on the transition of the Q output for a particular flip-flop. To illustrate this procedure, two sample entries are shown for the J_0 and the K_0 inputs to the least significant flip-flop (Q_0) in Figure 9–27.



▲ **FIGURE 9-27**

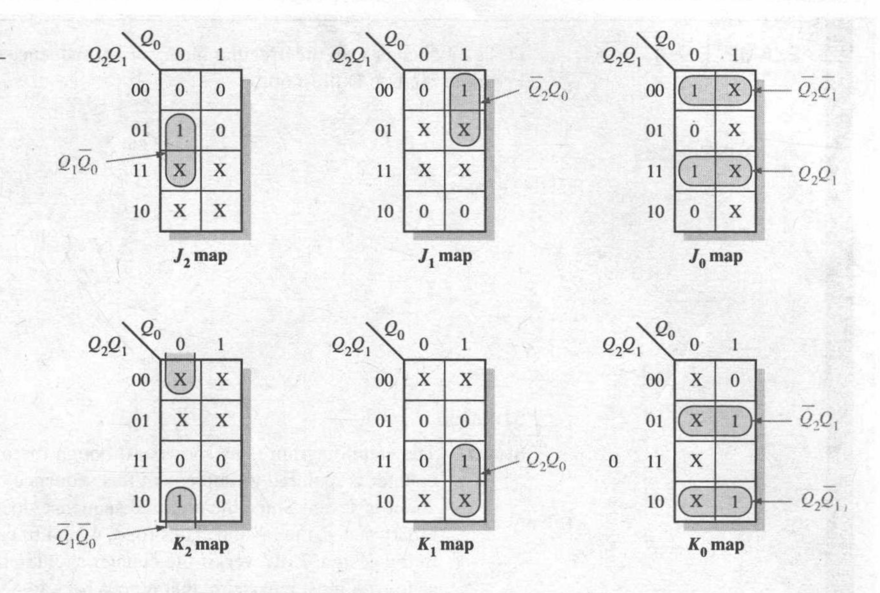
Examples of the mapping procedure for the counter sequence represented in Table 9–8 and Table 9–9.

The completed Karnaugh maps for all three flip-flops in the counter are shown in Figure 9–28. The cells are grouped as indicated and the corresponding Boolean expressions for each group are derived.

Step 5: Logic Expressions for Flip-Flop Inputs

From the Karnaugh maps of Figure 9–28 you obtain the following expressions for the J and K inputs of each flip-flop:

$$J_0 = Q_2Q_1 + \overline{Q}_2\overline{Q}_1 = \overline{Q_2 \oplus Q_1}$$
$$K_0 = Q_2\overline{Q}_1 + \overline{Q}_2Q_1 = Q_2 \oplus Q_1$$
$$J_1 = \overline{Q}_2Q_0$$
$$K_1 = Q_2Q_0$$
$$J_2 = Q_1\overline{Q}_0$$
$$K_2 = \overline{Q}_1\overline{Q}_0$$

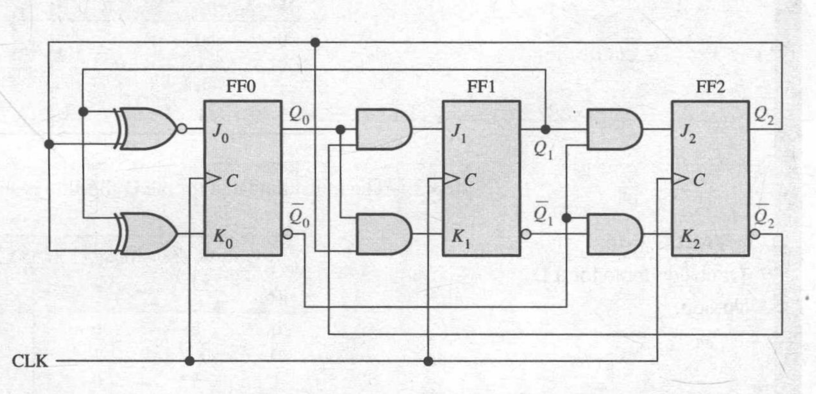


▶ FIGURE 9-28

Karnaugh maps for present-state J and K inputs.

Step 6: Counter Implementation

The final step is to implement the combinational logic from the expressions for the J and K inputs and connect the flip-flops to form the complete 3-bit Gray code counter as shown in Figure 9–29.



▶ FIGURE 9-29

Three-bit Gray code counter. Open file F09-29 to verify operation.

MultiSim

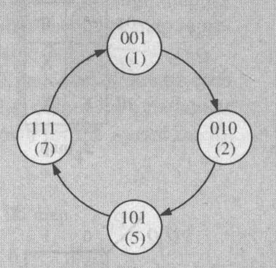
A summary of steps used in the design of the 3-bit Gray code counter follows. In general, these steps can be applied to any state machine.

1. Specify the counter sequence and draw a state diagram.
2. Derive a next-state table from the state diagram.
3. Develop a transition table showing the flip-flop inputs required for each transition. The transition table is always the same for a given type of flip-flop.
4. Transfer the J and K states from the transition table to Karnaugh maps. There is a Karnaugh map for each input of each flip-flop.
5. Group the Karnaugh map cells to generate and derive the logic expression for each flip-flop input.
6. Implement the expressions with combinational logic, and combine with the flip-flops to create the counter.

This procedure is now applied to the design of other synchronous counters in Examples 9–4 and 9–5.

EXAMPLE 9-4

Design a counter with the irregular binary count sequence shown in the state diagram of Figure 9–30. Use D flip-flops.



▶ FIGURE 9–30

Solution

Step 1: The state diagram is as shown. Although there are only four states, a 3-bit counter is required to implement this sequence because the maximum binary count is seven. Since the required sequence does not include all the possible binary states, the invalid states (0, 3, 4, and 6) can be treated as "don't cares" in the design. However, if the counter should erroneously get into an invalid state, you must make sure that it goes back to a valid state.

Step 2: The next-state table is developed from the state diagram and is given in Table 9–10.

▶ TABLE 9–10
Next-state table.

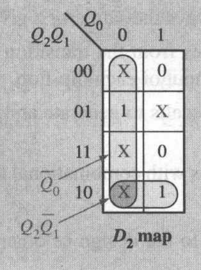
Present State			Next State		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	1	0	1	0
0	1	0	1	0	1
1	0	1	1	1	1
1	1	1	0	0	1

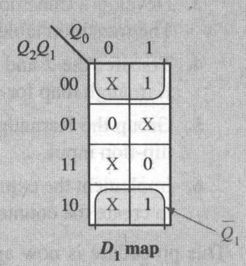
Step 3: The transition table for the D flip-flop is shown in Table 9–11.

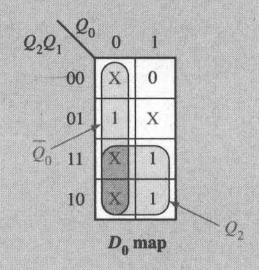
▶ TABLE 9–11
Transition table for a D flip-flop.

Output Transitions			Flip-Flop Input
Q_N		Q_{N+1}	D
0	⟶	0	0
0	⟶	1	1
1	⟶	0	0
1	⟶	1	1

Step 4: The *D* inputs are plotted on the present-state Karnaugh maps in Figure 9–31. Also "don't cares" can be placed in the cells corresponding to the invalid states of 000, 011, 100, and 110, as indicated by the red Xs.







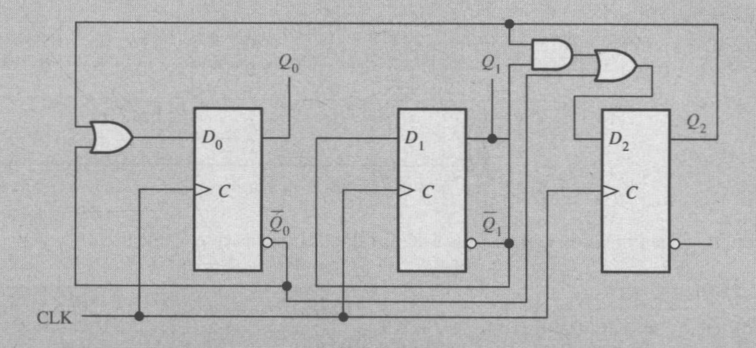
▶ FIGURE 9–31

Step 5: Group the 1s, taking advantage of as many of the "don't care" states as possible for maximum simplification, as shown in Figure 9–31. The expression for each D input taken from the maps is as follows:

$$D_0 = \overline{Q}_0 + Q_2$$
$$D_1 = \overline{Q}_1$$
$$D_2 = \overline{Q}_0 + Q_2\overline{Q}_1$$

Step 6: The implementation of the counter is shown in Figure 9–32.

FIGURE 9–32



An analysis shows that if the counter, by accident, gets into one of the invalid states (0, 3, 4, 6), it will always return to a valid state according to the following sequences: 0 → 3 → 4 → 7, and 6 → 1.

Related Problem

Verify the analysis that proves the counter will always return (eventually) to a valid state from an invalid state.

EXAMPLE 9-5

Develop a synchronous 3-bit up/down counter with a Gray code sequence using J-K flip-flops. The counter should count up when an UP/$\overline{\text{DOWN}}$ control input is 1 and count down when the control input is 0.

Solution

Step 1: The state diagram is shown in Figure 9–33. The 1 or 0 beside each arrow indicates the state of the UP/$\overline{\text{DOWN}}$ control input, Y.

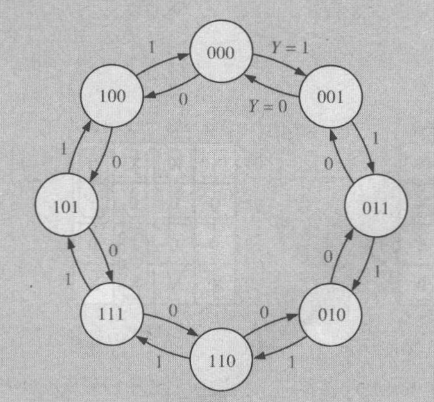


FIGURE 9–33
State diagram for a 3-bit up/down Gray code counter.

Step 2: The next-state table is derived from the state diagram and is shown in Table 9–12. Notice that for each present state there are two possible next states, depending on the UP/$\overline{\text{DOWN}}$ control variable, Y.

TABLE 9–12

Next-state table for 3-bit up/down Gray code counter.

Present State			Next State					
			$Y = 0$ (DOWN)			$Y = 1$ (UP)		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	1	0	0	0	0	1
0	0	1	0	0	0	0	1	1
0	1	1	0	0	1	0	1	0
0	1	0	0	1	1	1	1	0
1	1	0	0	1	0	1	1	1
1	1	1	1	1	0	1	0	1
1	0	1	1	1	1	1	0	0
1	0	0	1	0	1	0	0	0

Y = UP/$\overline{\text{DOWN}}$ control input.

Step 3: The transition table for the J-K flip-flops is repeated in Table 9–13.

TABLE 9–13

Transition table for a J-K flip-flop.

Output Transitions			Flip-Flop Inputs	
Q_N		Q_{N+1}	J	K
0	⟶	0	0	X
0	⟶	1	1	X
1	⟶	0	X	1
1	⟶	1	X	0

Step 4: The Karnaugh maps for the J and K inputs of the flip-flops are shown in Figure 9–34. The UP/$\overline{\text{DOWN}}$ control input, Y, is considered one of the state variables along with Q_0, Q_1, and Q_2. Using the next-state table, the information in the "Flip-Flop Inputs" column of Table 9–13 is transferred onto the maps as indicated for each present state of the counter.

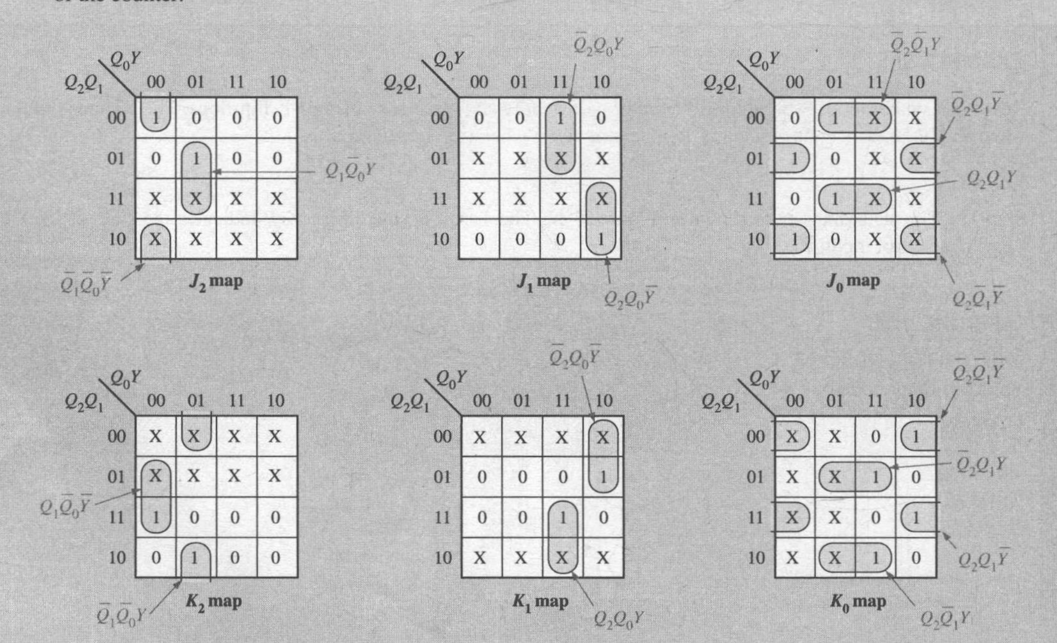


FIGURE 9–34

J and K maps for Table 9–12. The UP/$\overline{\text{DOWN}}$ control input, Y, is treated as a fourth variable.

Step 5: The 1s are combined in the largest possible groupings, with "don't cares" (Xs) used where possible. The groups are factored, and the expressions for the J and K inputs are as follows:

$$J_0 = Q_2Q_1Y + Q_2\overline{Q}_1\overline{Y} + \overline{Q}_2\overline{Q}_1Y + \overline{Q}_2Q_1\overline{Y} \qquad K_0 = \overline{Q}_2\overline{Q}_1\overline{Y} + \overline{Q}_2Q_1Y + Q_2\overline{Q}_1Y + Q_2Q_1\overline{Y}$$
$$J_1 = \overline{Q}_2Q_0Y + Q_2Q_0\overline{Y} \qquad K_1 = \overline{Q}_2Q_0\overline{Y} + Q_2Q_0Y$$
$$J_2 = Q_1\overline{Q}_0Y + \overline{Q}_1\overline{Q}_0\overline{Y} \qquad K_2 = Q_1\overline{Q}_0\overline{Y} + \overline{Q}_1\overline{Q}_0Y$$

Step 6: The J and K equations are implemented with combinational logic. This step is the Related Problem.

Related Problem

Specify the number of flip-flops, gates, and inverters that are required to implement the logic described in Step 5.

SECTION 9-5 CHECKUP

1. A flip-flop is presently in the RESET state and must go to the SET state on the next clock pulse. What must J and K be?
2. A flip-flop is presently in the SET state and must remain SET on the next clock pulse. What must J and K be?
3. A binary counter is in the $Q_3\overline{Q}_2Q_1\overline{Q}_0 = 1010$ state.
   (a) What is its next state?
   (b) What condition must exist on each flip-flop input to ensure that it goes to the proper next state on the clock pulse?

9-6 Cascaded Counters

Asynchronous Cascading

An example of two asynchronous counters connected in cascade is shown in Figure 9–35 for a 2-bit and a 3-bit ripple counter. The timing diagram is shown in Figure 9–36. Notice that the final output of the modulus-8 counter, Q_4, occurs once for every 32 input clock pulses. The overall modulus of the two cascaded counters is $4 \times 8 = 32$; that is, they act as a divide-by-32 counter.

The overall modulus of cascaded counters is equal to the product of the individual moduli.

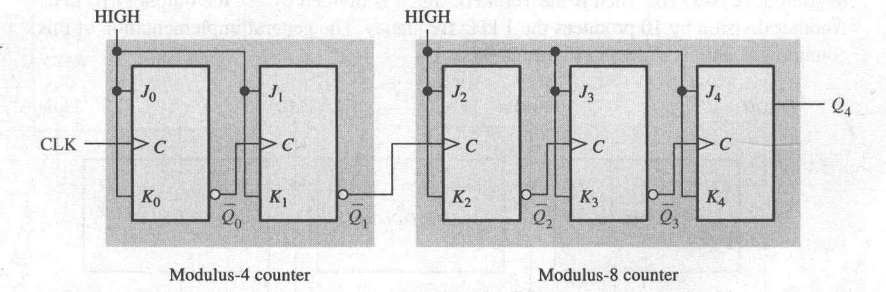


FIGURE 9–35 Two cascaded asynchronous counters (all J and K inputs are HIGH).

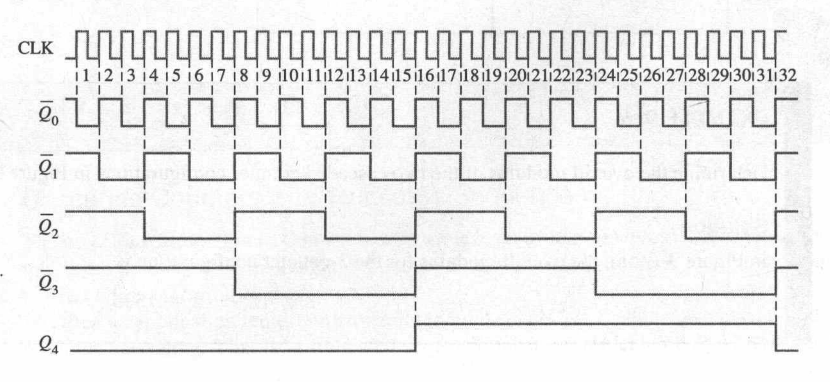


FIGURE 9–36 Timing diagram for the cascaded counter configuration of Figure 9–35.

Synchronous Cascading

When operating synchronous counters in a cascaded configuration, it is necessary to use the count enable and the terminal count functions to achieve higher-modulus operation. On some devices the count enable is labeled simply *CTEN* (or some other designation such as *G*), and terminal count (*TC*) is analogous to ripple clock output (*RCO*) on some IC counters.

Figure 9–37 shows two decade counters connected in cascade. The terminal count (*TC*) output of counter 1 is connected to the count enable (*CTEN*) input of counter 2. Counter 2 is inhibited by the LOW on its *CTEN* input until counter 1 reaches its last, or terminal, state and its terminal count output goes HIGH. This HIGH now enables counter 2, so that when the first clock pulse after counter 1 reaches its terminal count (CLK10), counter 2 goes from its initial state to its second state. Upon completion of the entire second cycle of counter 1 (when counter 1 reaches terminal count the second time), counter 2 is again enabled and advances to its next state. This sequence continues. Since these are decade counters, counter 1 must go through ten complete cycles before counter 2 completes its first cycle. In other words, for every ten cycles of counter 1, counter 2 goes through one cycle. Thus, counter 2 will complete one cycle after one hundred clock pulses. The overall modulus of these two cascaded counters is $10 \times 10 = 100$.

InfoNote

The time stamp counter (TSC), mentioned in the last InfoNote, is a 64-bit counter. It is interesting to observe that if this counter (or any full-modulus 64-bit counter) is clocked at a frequency of 1 GHz, it will take 583 years for it to go through all of its states and reach its terminal count. In contrast, a 32-bit full-modulus counter will exhaust all of its states in approximately 4.3 seconds when clocked at 1 GHz. The difference is astounding.

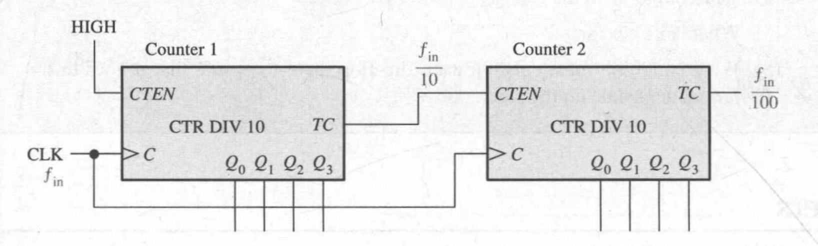


FIGURE 9–37 A modulus-100 counter using two cascaded decade counters.

When viewed as a frequency divider, the circuit of Figure 9–37 divides the input clock frequency by 100. Cascaded counters are often used to divide a high-frequency clock signal to obtain highly accurate pulse frequencies. Cascaded counter configurations used for such purposes are sometimes called *countdown chains.* For example, suppose that you have a basic clock frequency of 1 MHz and you wish to obtain 100 kHz, 10 kHz, and 1 kHz; a series of cascaded decade counters can be used. If the 1 MHz signal is divided by 10, the output is 100 kHz. Then if the 100 kHz signal is divided by 10, the output is 10 kHz. Another division by 10 produces the 1 kHz frequency. The general implementation of this countdown chain is shown in Figure 9–38.

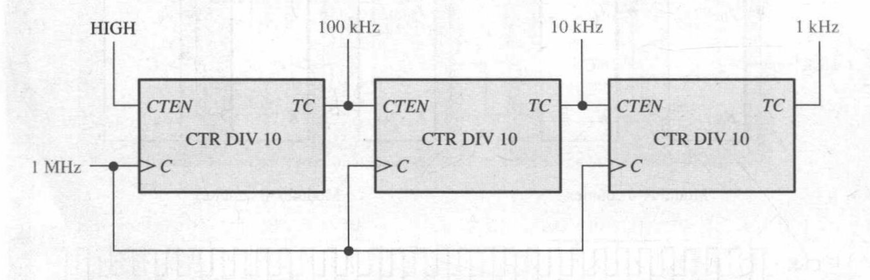


FIGURE 9–38 Three cascaded decade counters forming a divide-by-1000 frequency divider with intermediate divide-by-10 and divide-by-100 outputs.

EXAMPLE 9-6

Determine the overall modulus of the two cascaded counter configurations in Figure 9–39.

Solution

In Figure 9–39(a), the overall modulus for the 3-counter configuration is

$$8 \times 12 \times 16 = \mathbf{1536}$$

In Figure 9–39(b), the overall modulus for the 4-counter configuration is

$$10 \times 4 \times 7 \times 5 = \mathbf{1400}$$

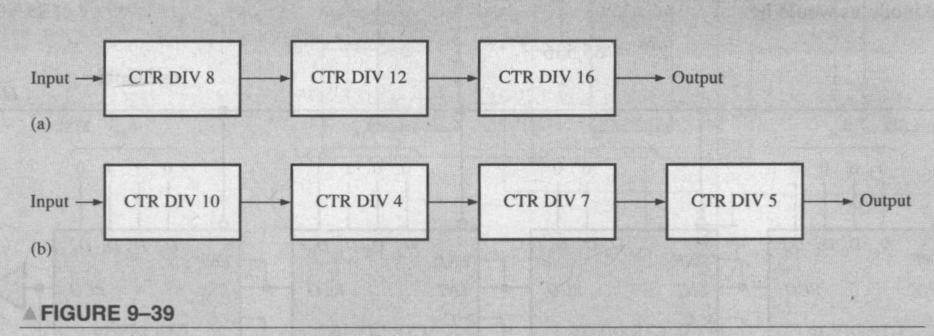


FIGURE 9–39

Related Problem

How many cascaded decade counters are required to divide a clock frequency by 100,000?

EXAMPLE 9-7

Use 74HC190 up/down decade counters connected in the UP mode to obtain a 10 kHz waveform from a 1 MHz clock. Show the logic diagram.

Solution

To obtain 10 kHz from a 1 MHz clock requires a division factor of 100. Two 74HC190 counters must be cascaded as shown in Figure 9–40. The left counter produces a terminal count (*MAX/MIN*) pulse for every 10 clock pulses. The right counter produces a terminal count (*MAX/MIN*) pulse for every 100 clock pulses.

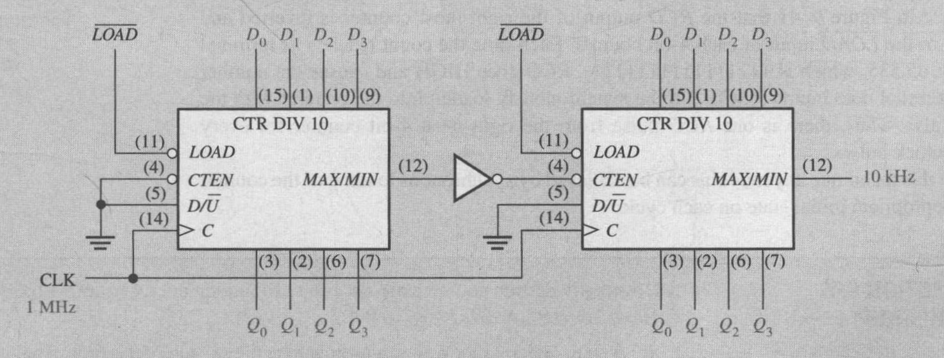


FIGURE 9–40

A divide-by-100 counter using two 74HC190 up/down decade counters connected for the up sequence.

Related Problem

Determine the frequency of the waveform at the Q_0 output of the second counter (the one on the right) in Figure 9–40.

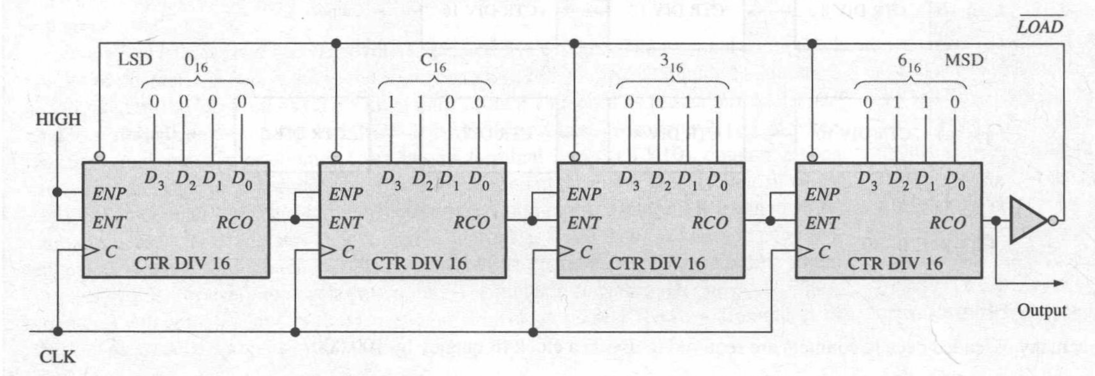
Cascaded Counters with Truncated Sequences

The preceding discussion has shown how to achieve an overall modulus (divide-by-factor) that is the product of the individual moduli of all the cascaded counters. This can be considered *full-modulus cascading*.

Often an application requires an overall modulus that is less than that achieved by full-modulus cascading. That is, a truncated sequence must be implemented with cascaded

counters. To illustrate this method, we will use the cascaded counter configuration in Figure 9–41. This particular circuit uses four 74HC161 4-bit synchronous binary counters. If these four counters (sixteen bits total) were cascaded in a full-modulus arrangement, the modulus would be

$$2^{16} = 65{,}536$$



▲ **FIGURE 9–41**

A divide-by-40,000 counter using 74HC161 4-bit binary counters. Note that each of the parallel data inputs is shown in binary order (the right-most bit D_0 is the LSB in each counter).

Let's assume that a certain application requires a divide-by-40,000 counter (modulus 40,000). The difference between 65,536 and 40,000 is 25,536, which is the number of states that must be *deleted* from the full-modulus sequence. The technique used in the circuit of Figure 9–41 is to preset the cascaded counter to 25,536 (63C0 in hexadecimal) each time it recycles, so that it will count from 25,536 up to 65,535 on each full cycle. Therefore, each full cycle of the counter consists of 40,000 states.

Notice in Figure 9–41 that the *RCO* output of the right-most counter is inverted and applied to the $\overline{LOAD}$ input of each 4-bit counter. Each time the count reaches its terminal value of 65,535, which is 1111111111111111_2, *RCO* goes HIGH and causes the number on the parallel data inputs ($63C0_{16}$) to be synchronously loaded into the counter with the clock pulse. Thus, there is one *RCO* pulse from the right-most 4-bit counter for every 40,000 clock pulses.

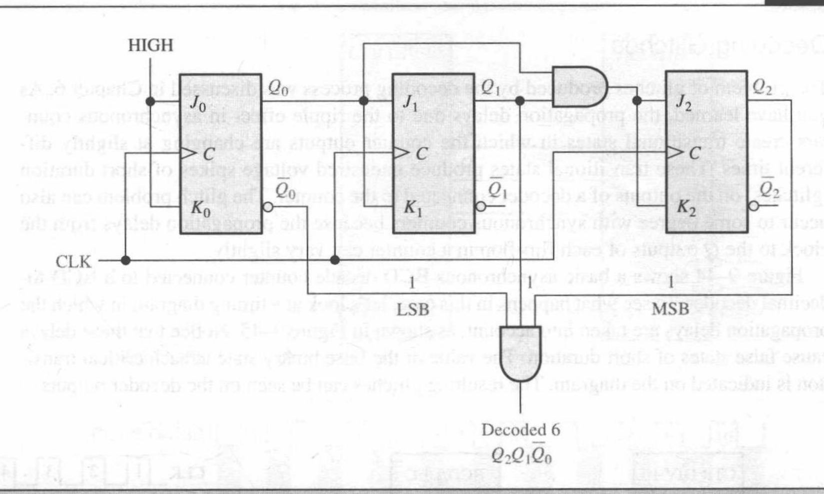
With this technique any modulus can be achieved by synchronous loading of the counter to the appropriate initial state on each cycle.

SECTION 9-6 CHECKUP

1. How many decade counters are necessary to implement a divide-by-1000 (modulus-1000) counter? A divide-by-10,000?
2. Show with general block diagrams how to achieve each of the following, using a flip-flop, a decade counter, and a 4-bit binary counter, or any combination of these:
   (a) Divide-by-20 counter
   (b) Divide-by-32 counter
   (c) Divide-by-160 counter
   (d) Divide-by-320 counter

9-7 Counter Decoding

Suppose that you wish to decode binary state 6 (110) of a 3-bit binary counter. When $Q_2 = 1$, $Q_1 = 1$, and $Q_0 = 0$, a HIGH appears on the output of the decoding gate, indicating that the counter is at state 6. This can be done as shown in Figure 9–42. This is called *active-HIGH decoding.* Replacing the AND gate with a NAND gate provides active-LOW decoding.



▶ **FIGURE 9–42**

Decoding of state 6 (110). Open file F09-42 to verify operation.

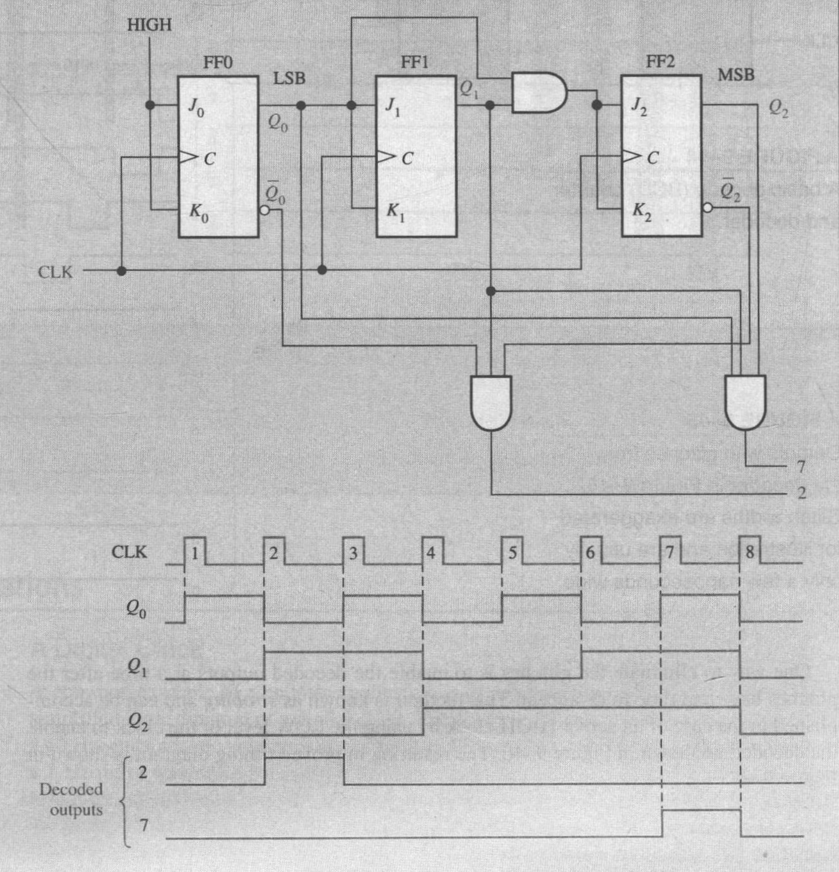
MultiSim

EXAMPLE 9-8

Implement the decoding of binary state 2 and binary state 7 of a 3-bit synchronous counter. Show the entire counter timing diagram and the output waveforms of the decoding gates. Binary 2 = $\overline{Q}_2Q_1\overline{Q}_0$ and binary 7 = $Q_2Q_1Q_0$.

Solution

See Figure 9–43. The 3-bit counter was originally discussed in Section 9–3 (Figure 9–15).



▶ **FIGURE 9–43**

A 3-bit counter with active-HIGH decoding of count 2 and count 7. Open file F09-43 to verify operation.

MultiSim

Related Problem

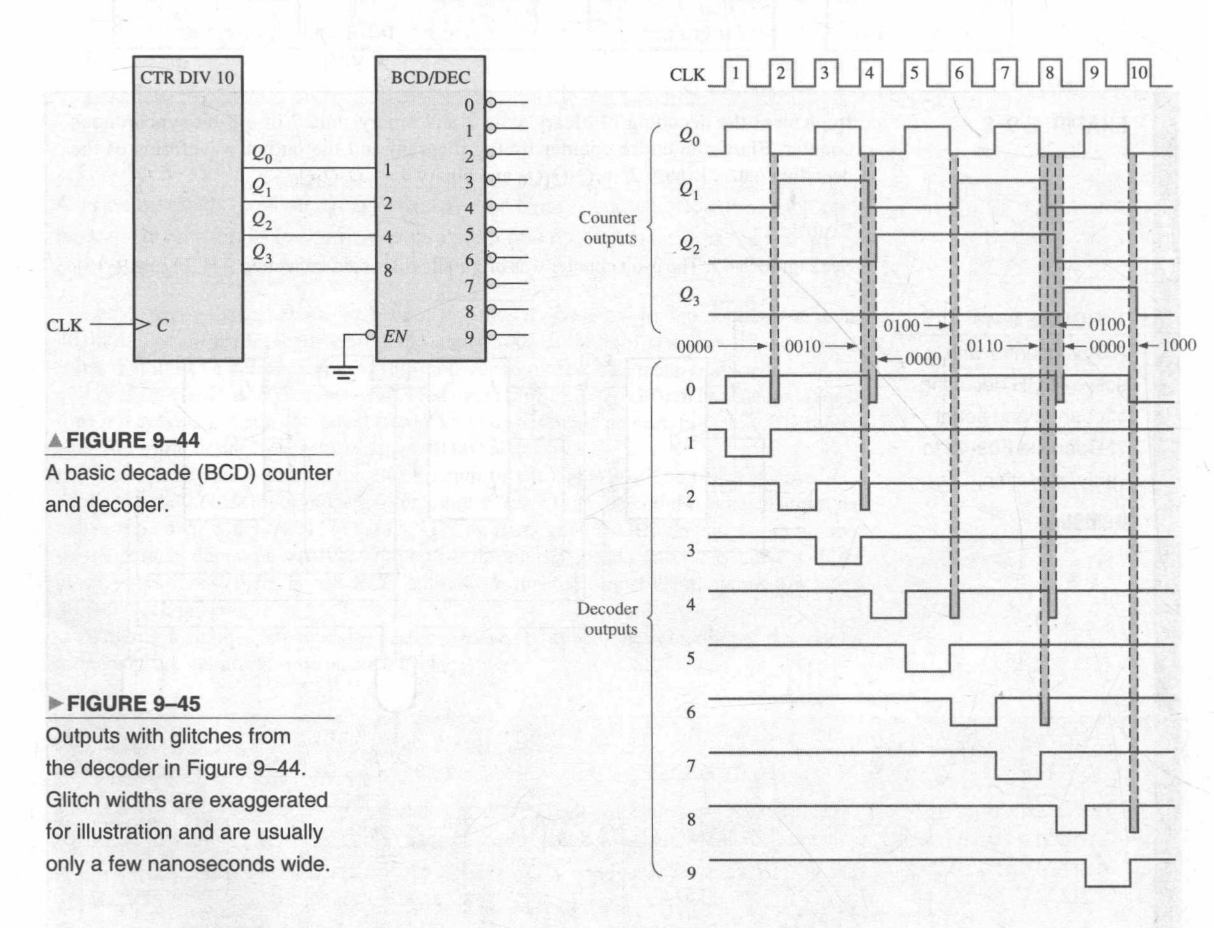
Show the logic for decoding state 5 in the 3-bit counter.

Decoding Glitches

A glitch is an unwanted spike of voltage.

The problem of glitches produced by the decoding process was discussed in Chapter 6. As you have learned, the propagation delays due to the ripple effect in asynchronous counters create transitional states in which the counter outputs are changing at slightly different times. These transitional states produce undesired voltage spikes of short duration (glitches) on the outputs of a decoder connected to the counter. The glitch problem can also occur to some degree with synchronous counters because the propagation delays from the clock to the Q outputs of each flip-flop in a counter can vary slightly.

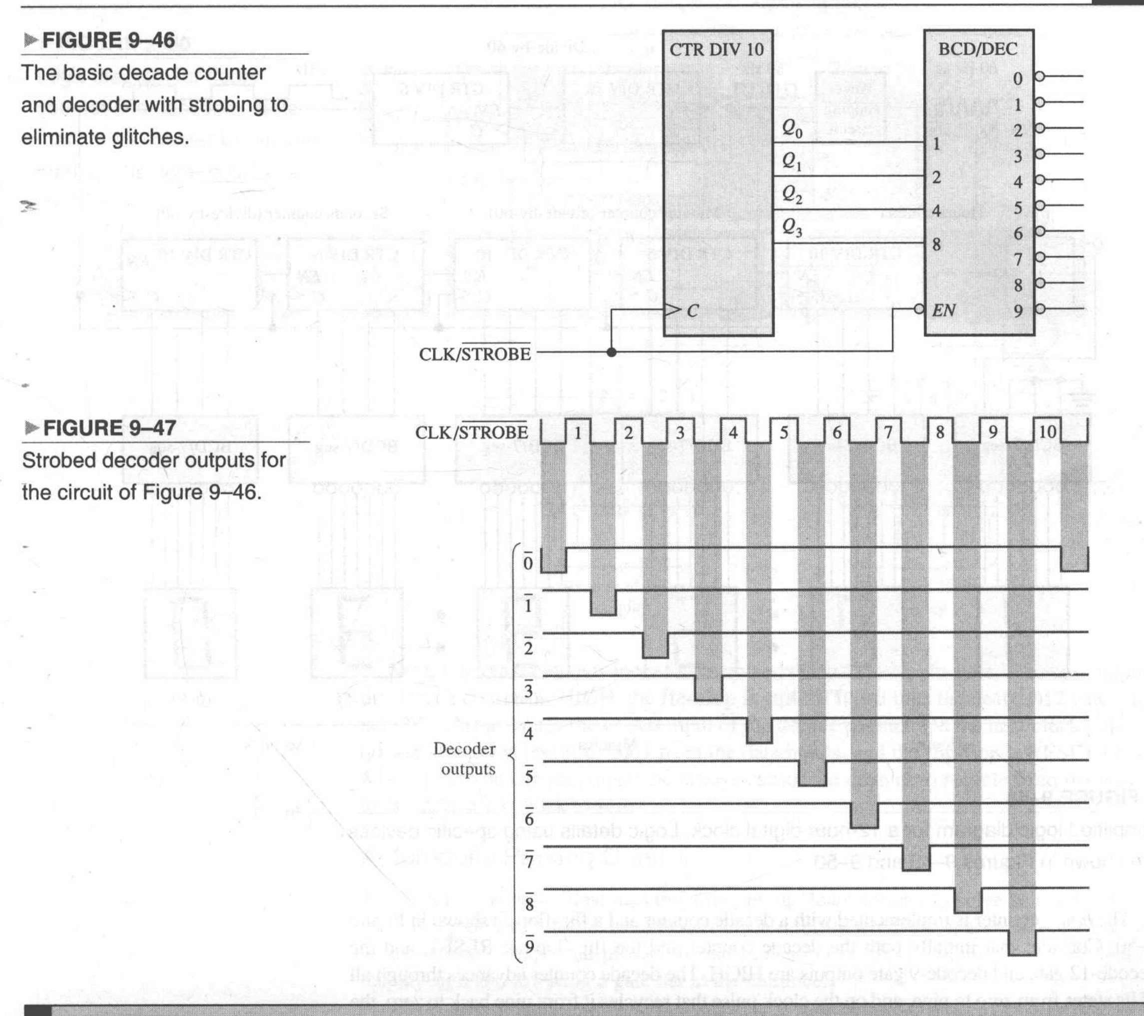
Figure 9–44 shows a basic asynchronous BCD decade counter connected to a BCD-to-decimal decoder. To see what happens in this case, let's look at a timing diagram in which the propagation delays are taken into account, as shown in Figure 9–45. Notice that these delays cause false states of short duration. The value of the false binary state at each critical transition is indicated on the diagram. The resulting glitches can be seen on the decoder outputs.



▲ **FIGURE 9–44**
A basic decade (BCD) counter and decoder.

► **FIGURE 9–45**
Outputs with glitches from the decoder in Figure 9–44. Glitch widths are exaggerated for illustration and are usually only a few nanoseconds wide.

One way to eliminate the glitches is to enable the decoded outputs at a time after the glitches have had time to disappear. This method is known as *strobing* and can be accomplished in the case of an active-HIGH clock by using the LOW level of the clock to enable the decoder, as shown in Figure 9–46. The resulting improved timing diagram is shown in Figure 9–47.



▶ **FIGURE 9–46**
The basic decade counter and decoder with strobing to eliminate glitches.

▶ **FIGURE 9–47**
Strobed decoder outputs for the circuit of Figure 9–46.

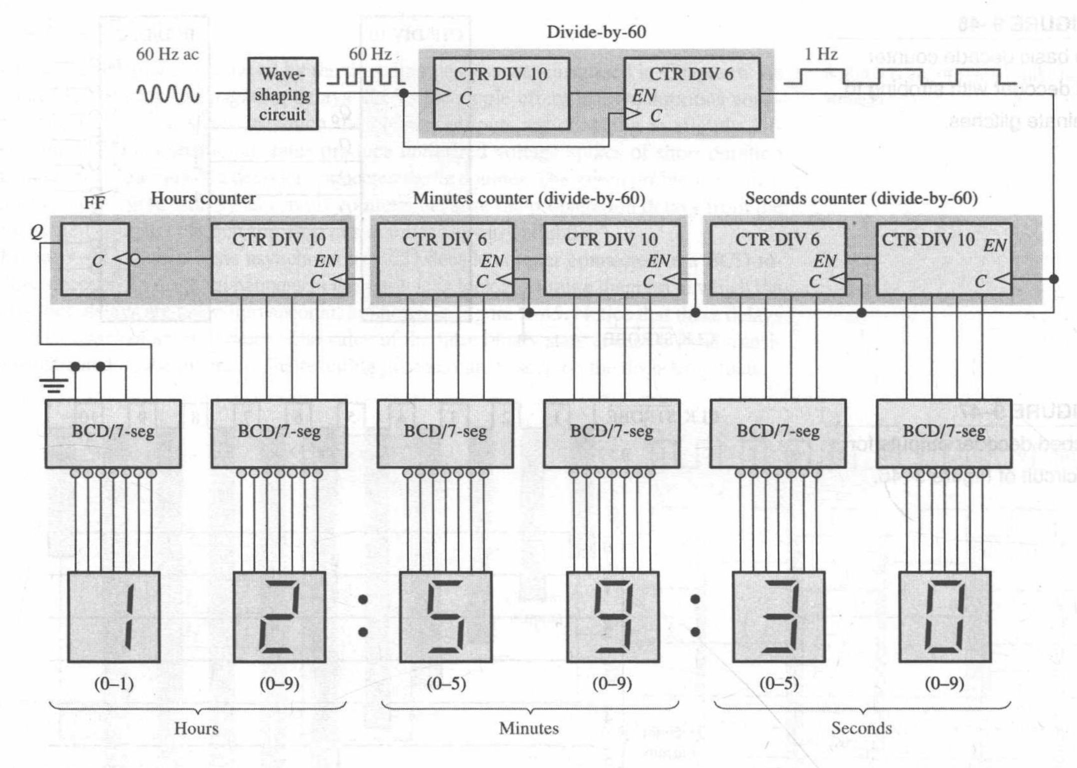
SECTION 9-7 CHECKUP

1. What transitional states are possible when a 4-bit asynchronous binary counter changes from
   (a) count 2 to count 3 (b) count 3 to count 4
   (c) count 10_{10} to count 11_{10} (d) count 15 to count 0

9-8 Counter Applications

A Digital Clock

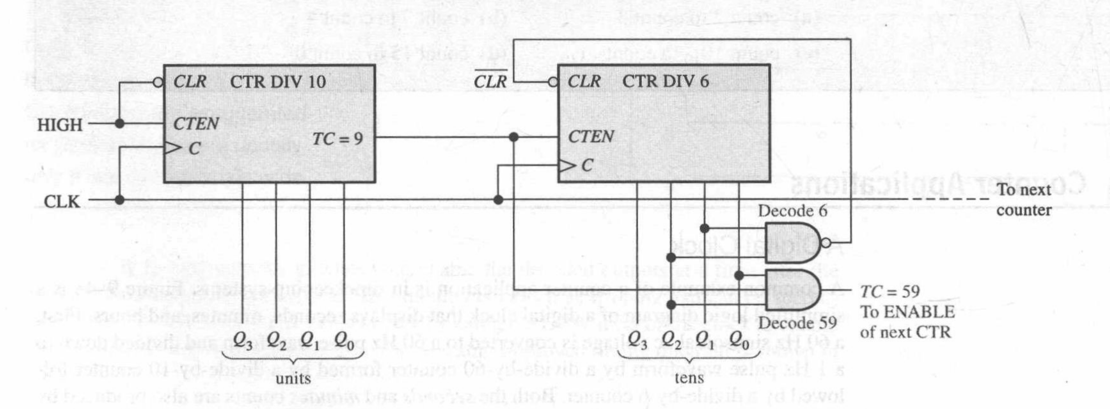
A common example of a counter application is in timekeeping systems. Figure 9–48 is a simplified logic diagram of a digital clock that displays seconds, minutes, and hours. First, a 60 Hz sinusoidal ac voltage is converted to a 60 Hz pulse waveform and divided down to a 1 Hz pulse waveform by a divide-by-60 counter formed by a divide-by-10 counter followed by a divide-by-6 counter. Both the *seconds* and *minutes* counts are also produced by divide-by-60 counters, the details of which are shown in Figure 9–49. These counters count from 0 to 59 and then recycle to 0; synchronous decade counters are used in this particular implementation. Notice that the divide-by-6 portion is formed with a decade counter with a truncated sequence achieved by using the decoder count 6 to asynchronously clear the counter. The terminal count, 59, is also decoded to enable the next counter in the chain.



▲ **FIGURE 9–48**

Simplified logic diagram for a 12-hour digital clock. Logic details using specific devices are shown in Figures 9–49 and 9–50.

The *hours* counter is implemented with a decade counter and a flip-flop as shown in Figure 9–50. Consider that initially both the decade counter and the flip-flop are RESET, and the decode-12 gate and decode-9 gate outputs are HIGH. The decade counter advances through all of its states from zero to nine, and on the clock pulse that recycles it from nine back to zero, the flip-flop goes to the SET state ($J = 1$, $K = 0$). This illuminates a 1 on the tens-of-hours display. The total count is now ten (the decade counter is in the zero state and the flip-flop is SET).

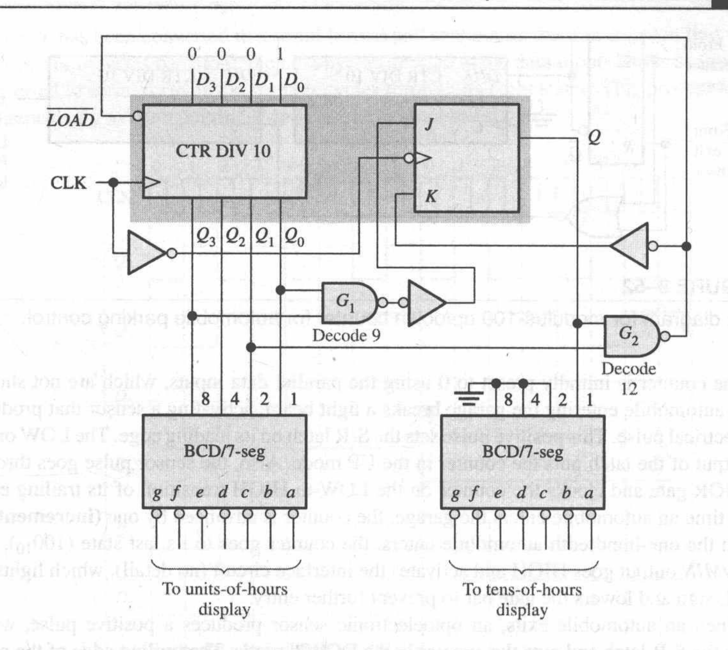


▲ **FIGURE 9–49**

Logic diagram of typical divide-by-60 counter using synchronous decade counters. Note that the outputs are in binary order (the right-most bit is the LSB).

FIGURE 9–50

Logic diagram for hours counter and decoders. Note that on the counter inputs and outputs, the right-most bit is the LSB.



Next, the total count advances to eleven and then to twelve. In state 12 the Q_2 output of the decade counter is HIGH, the flip-flop is still SET, and thus the decode-12 gate output is LOW. This activates the $\overline{LOAD}$ input of the decade counter. On the next clock pulse, the decade counter is preset to 0001 from the data inputs, and the flip-flop is RESET ($J = 0$, $K = 1$). As you can see, this logic always causes the counter to recycle from twelve back to one rather than back to zero.

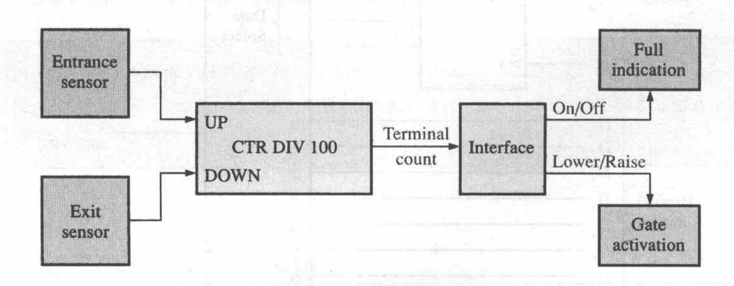
Automobile Parking Control

This counter example illustrates the use of an up/down counter to solve an everyday problem. The problem is to devise a means of monitoring available spaces in a one-hundred-space parking garage and provide for an indication of a full condition by illuminating a display sign and lowering a gate bar at the entrance.

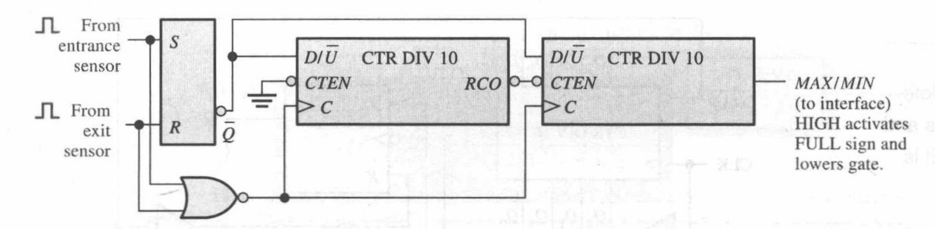
A system that solves this problem consists of optoelectronic sensors at the entrance and exit of the garage, an up/down counter and associated circuitry, and an interface circuit that uses the counter output to turn the FULL sign on or off as required and lower or raise the gate bar at the entrance. A general block diagram of this system is shown in Figure 9–51.

FIGURE 9–51

Functional block diagram for parking garage control.



A logic diagram of the up/down counter is shown in Figure 9–52. It consists of two cascaded up/down decade counters. The operation is described in the following paragraphs.



▲ **FIGURE 9–52**

Logic diagram for modulus-100 up/down counter for automobile parking control.

The counter is initially preset to 0 using the parallel data inputs, which are not shown. Each automobile entering the garage breaks a light beam, activating a sensor that produces an electrical pulse. This positive pulse sets the S-R latch on its leading edge. The LOW on the $\overline{Q}$ output of the latch puts the counter in the UP mode. Also, the sensor pulse goes through the NOR gate and clocks the counter on the LOW-to-HIGH transition of its trailing edge. Each time an automobile enters the garage, the counter is advanced by one **(incremented)**. When the one-hundredth automobile enters, the counter goes to its last state (100_{10}). The *MAX/MIN* output goes HIGH and activates the interface circuit (no detail), which lights the FULL sign and lowers the gate bar to prevent further entry.

Incrementing a counter increases its count by one.

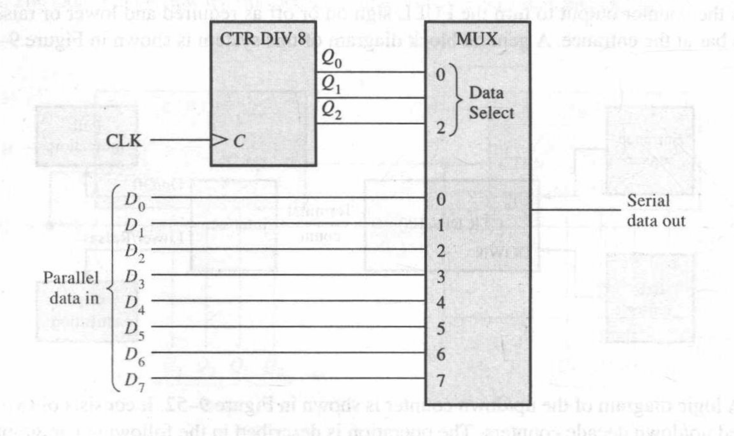
When an automobile exits, an optoelectronic sensor produces a positive pulse, which resets the S-R latch and puts the counter in the DOWN mode. The trailing edge of the clock decreases the count by one **(decremented)**. If the garage is full and an automobile leaves, the *MAX/MIN* output of the counter goes LOW, turning off the FULL sign and raising the gate.

Decrementing a counter decreases it count by one.

Parallel-to-Serial Data Conversion (Multiplexing)

A simplified example of data transmission using multiplexing and demultiplexing techniques was introduced in Chapter 6. Essentially, the parallel data bits on the multiplexer inputs are converted to serial data bits on the single transmission line. A group of bits appearing simultaneously on parallel lines is called *parallel data.* A group of bits appearing on a single line in a time sequence is called *serial data.*

Parallel-to-serial conversion is normally accomplished by the use of a counter to provide a binary sequence for the data-select inputs of a data selector/multiplexer, as illustrated in Figure 9–53. The Q outputs of the modulus-8 counter are connected to the data-select inputs of an 8-bit multiplexer.

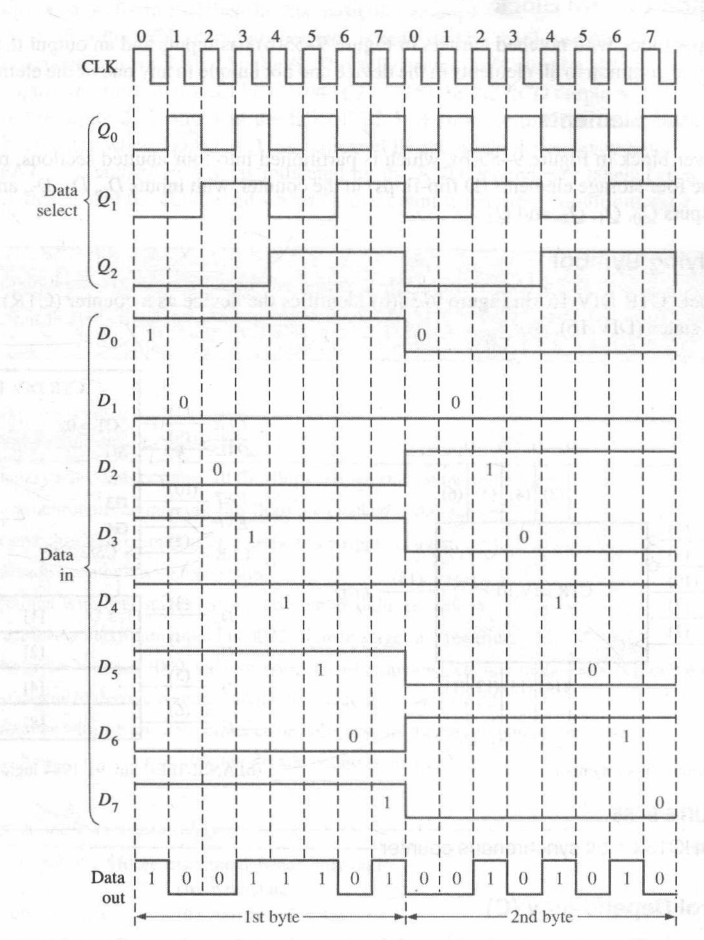


◀ **FIGURE 9–53**

Parallel-to-serial data conversion logic.

Figure 9–54 is a timing diagram illustrating the operation of this circuit. The first byte (eight-bit group) of parallel data is applied to the multiplexer inputs. As the counter goes through a binary sequence from zero to seven, each bit, beginning with D_0, is sequentially selected and passed through the multiplexer to the output line. After eight clock pulses the

data byte has been converted to a serial format and sent out on the transmission line. When the counter recycles back to 0, the next byte is applied to the data inputs and is sequentially converted to serial form as the counter cycles through its eight states. This process continues repeatedly as each parallel byte is converted to a serial byte.



▲ **FIGURE 9–54**

Example of parallel-to-serial conversion timing for the circuit in Figure 9–53.

InfoNote

Computers contain an internal counter that can be programmed for various frequencies and tone durations, thus producing "music." To select a particular tone, the programmed instruction selects a divisor that is sent to the counter. The divisor sets the counter up to divide the basic peripheral clock frequency to produce an audio tone. The duration of a tone can also be set by a programmed instruction; thus, a basic counter is used to produce melodies by controlling the frequency and duration of tones.

SECTION 9-8 CHECKUP

1. Explain the purpose of each NAND gate in Figure 9–50.
2. Identify the two recycle conditions for the hours counter in Figure 9–48, and explain the reason for each.

9-9 Logic Symbols with Dependency Notation

Dependency notation is fundamental to the ANSI/IEEE standard. Dependency notation is used in conjunction with the logic symbols to specify the relationships of inputs and outputs so that the logical operation of a given device can be determined entirely from its logic symbol without a prior knowledge of the details of its internal structure and without a detailed logic diagram for reference. This coverage of a specific logic symbol with dependency notation is intended to aid in the interpretation of other such symbols that you may encounter in the future.

The 74HC163 4-bit synchronous binary counter is used for illustration. For comparison, Figure 9–55 shows a traditional block symbol and the ANSI/IEEE symbol with dependency notation. Basic descriptions of the symbol and the dependency notation follow.

Common Control Block

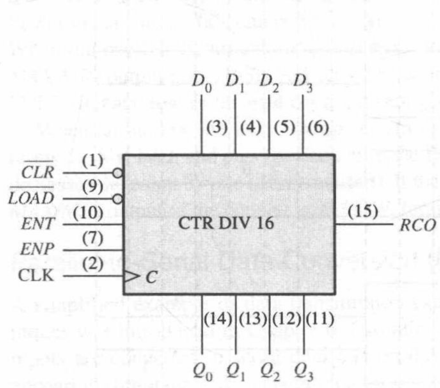
The upper block with notched corners in Figure 9–55(b) has inputs and an output that are considered common to all elements in the device and not unique to any one of the elements.

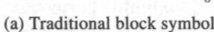
Individual Elements

The lower block in Figure 9–55(b), which is partitioned into four abutted sections, represents the four storage elements (D flip-flops) in the counter, with inputs D_0, D_1, D_2, and D_3 and outputs Q_0, Q_1, Q_2, and Q_3.

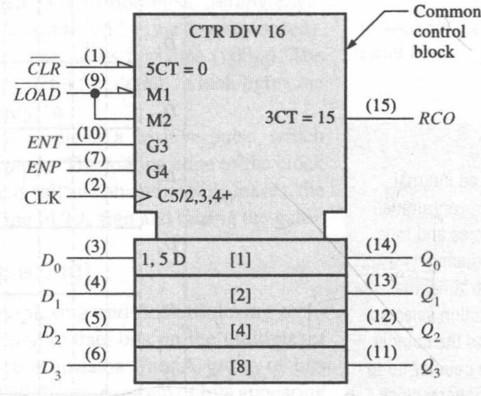
Qualifying Symbol

The label "CTR DIV 16" in Figure 9–55(b) identifies the device as a counter (CTR) with sixteen states (DIV 16).





(a) Traditional block symbol



(b) ANSI/IEEE Std. 91-1984 logic symbol

▲ **FIGURE 9–55**

The 74HC163 4-bit synchronous counter.

Control Dependency (C)

As shown in Figure 9–55(b), the letter *C* denotes control dependency. Control inputs usually enable or disable the data inputs (*D*, *J*, *K*, *S*, and *R*) of a storage element. The *C* input is usually the clock input. In this case the digit 5 following *C* (C5/2,3,4+) indicates that the inputs labeled with a 5 prefix are dependent on the clock (synchronous with the clock). For example, 5CT = 0 on the $\overline{CLR}$ input indicates that the clear function is dependent on the clock; that is, it is a synchronous clear. When the $\overline{CLR}$ input is LOW (0), the counter is reset to zero ($CT = 0$) on the triggering edge of the clock pulse. Also, the 5 D label at the input of storage element [1] indicates that the data storage is dependent on (synchronous with) the clock. All labels in the [1] storage element apply to the [2], [4], and [8] elements below it since they are not labeled differently.

Mode Dependency (M)

As shown in Figure 9–55(b), the letter *M* denotes mode dependency. This label is used to indicate how the functions of various inputs or outputs depend on the mode in which the device is operating. In this case the device has two modes of operation. When the $\overline{LOAD}$ input is LOW (0), as indicated by the triangle input, the counter is in a preset mode (M1) in which the input data (D_0, D_1, D_2, and D_3) are synchronously loaded into the four flip-flops. The digit 1 following *M* in M1 and the 1 in the label 1, 5 D show a dependency relationship and indicate that input data are stored only when the device is in the preset mode (M1), in which $\overline{LOAD} = 0$. When the $\overline{LOAD}$ input is HIGH (1), the counter advances through its normal binary sequence, as indicated by M2 and the 2 in C5/2,3,4+.

AND Dependency (G)

As shown in Figure 9–55(b), the letter *G* denotes AND dependency, indicating that an input designated with *G* followed by a digit is ANDed with any other input or output having the same digit as a prefix in its label. In this particular example, the G3 at the *ENT* input and the 3CT = 15 at the *RCO* output are related, as indicated by the 3, and that relationship is an AND dependency, indicated by the *G*. This tells us that *ENT* must be HIGH (no triangle on the input) *and* the count must be fifteen (*CT* = 15) for the *RCO* output to be HIGH.

Also, the digits 2, 3, and 4 in the label C5/2,3,4+ indicate that the counter advances through its states when $\overline{LOAD} = 1$, as indicated by the mode dependency label M2, and when *ENT* = 1 and *ENP* = 1, as indicated by the AND dependency labels G3 and G4. The + indicates that the counter advances by one count when these conditions exist.

SECTION 9-9 CHECKUP

1. In dependency notation, what do the letters *C*, *M*, and *G* stand for?
2. By what letter is data storage denoted?

TRUE/FALSE QUIZ

Answers are at the end of the chapter.

1. In an asynchronous counter, all flip-flops change state at the same time.
2. In a synchronous counter, all flip-flops are clocked simultaneously.
3. An asynchronous counter is also known as a ripple counter.
4. A decade counter has sixteen states.
5. A counter with four stages has a maximum modulus of sixteen.
6. To achieve a maximum modulus of 32, sixteen stages are required.
7. If the present state is 1000, the next state of a 4-bit up/down counter in the DOWN mode is 0111.
8. Two cascaded decade counters divide the clock frequency by 20.
9. A counter with a truncated sequence has less than its maximum number of states.
10. To achieve a modulus of 100, ten decade counters are required.

SELF-TEST

Answers are at the end of the chapter.

1. The output of a Moore machine depends only on its
   (a) inputs **(b)** next state
   (c) present state **(d)** number of states
2. The output of a Mealy machine depends on its
   (a) inputs **(b)** next state
   (c) present state **(d)** answers (a) and (c)
3. Asynchronous counters are known as
   (a) ripple counters **(b)** multiple clock counters
   (c) decade counters **(d)** modulus counters
4. An asynchronous counter differs from a synchronous counter in
   (a) the number of states in its sequence **(b)** the method of clocking
   (c) the type of flip-flops used **(d)** the value of the modulus
5. The modulus of a counter is
   (a) the number of flip-flops
   (b) the actual number of states in its sequence
   (c) the number of times it recycles in a second
   (d) the maximum possible number of states
6. A 3-bit binary counter has a maximum modulus of
   (a) 3 **(b)** 6 **(c)** 8 **(d)** 16
7. A 4-bit binary counter has a maximum modulus of
   (a) 16 **(b)** 32 **(c)** 8 **(d)** 4
8. A modulus-12 counter must have
   (a) 12 flip-flops **(b)** 3 flip-flops
   (c) 4 flip-flops **(d)** synchronous clocking

9. Which one of the following is an example of a counter with a truncated modulus?
   (a) Modulus 8 (b) Modulus 14
   (c) Modulus 16 (d) Modulus 32
10. A 4-bit ripple counter consists of flip-flops that each have a propagation delay from clock to Q output of 12 ns. For the counter to recycle from 1111 to 0000, it takes a total of
   (a) 12 ns (b) 24 ns
   (c) 48 ns (d) 36 ns
11. A BCD counter is an example of
   (a) a full-modulus counter (b) a decade counter
   (c) a truncated-modulus counter (d) answers (b) and (c)
12. Which of the following is an invalid state in an 8421 BCD counter?
   (a) 1100 (b) 0010
   (c) 0101 (d) 1000
13. Three cascaded modulus-10 counters have an overall modulus of
   (a) 30 (b) 100
   (c) 1000 (d) 10,000
14. A 10 MHz clock frequency is applied to a cascaded counter consisting of a modulus-5 counter, a modulus-8 counter, and two modulus-10 counters. The lowest output frequency possible is
   (a) 10 kHz (b) 2.5 kHz
   (c) 5 kHz (d) 25 kHz
15. A 4-bit binary up/down counter is in the binary state of zero. The next state in the DOWN mode is
   (a) 0001 (b) 1111
   (c) 1000 (d) 1110
16. The terminal count of a modulus-13 binary counter is
   (a) 0000 (b) 1111
   (c) 1101 (d) 1100

PROBLEMS

Answers to odd-numbered problems are at the end of the book.

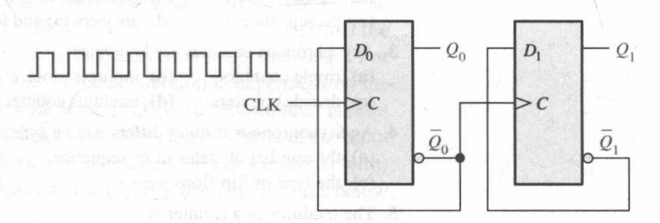
Section 9–1 Finite State Machines

1. Represent a decade counter with the terminal state decoded as a state machine. Identify the type and show the block diagram and the state diagram.
2. Identify the type of state machine for the traffic signal controller in Chapter 6. State the reason why it is the type you specified.

Section 9–2 Asynchronous Counters

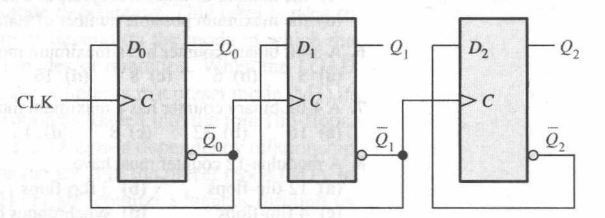
3. For the ripple counter shown in Figure 9–56, show the complete timing diagram for eight clock pulses, showing the clock, Q_0, and Q_1 waveforms.

▶ FIGURE 9–56



4. For the ripple counter in Figure 9–57, show the complete timing diagram for sixteen clock pulses. Show the clock, Q_0, Q_1, and Q_2 waveforms.

▶ FIGURE 9–57



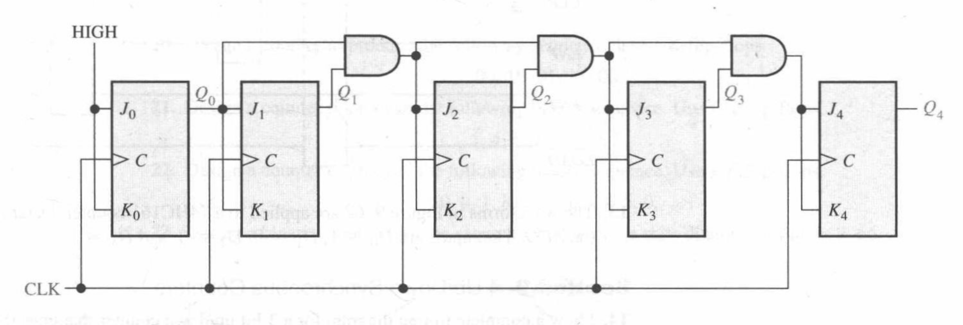
5. In the counter of Problem 4, assume that each flip-flop has a propagation delay from the triggering edge of the clock to a change in the Q output of 8 ns. Determine the worst-case (longest) delay time from a clock pulse to the arrival of the counter in a given state. Specify the state or states for which this worst-case delay occurs.

6. Show how to connect a 74HC93 4-bit asynchronous counter for each of the following moduli:
(a) 9 **(b)** 11 **(c)** 13 **(d)** 14 **(e)** 15

Section 9–3 Synchronous Counters

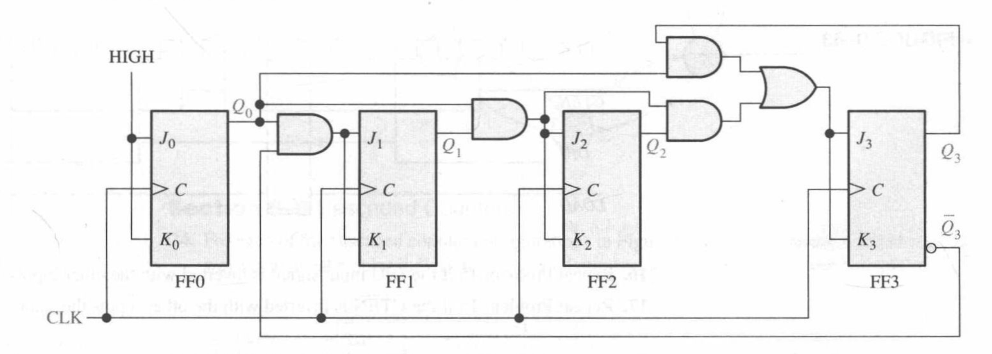
7. If the counter of Problem 5 were synchronous rather than asynchronous, what would be the longest delay time?

8. Show the complete timing diagram for the 5-stage synchronous binary counter in Figure 9–58. Verify that the waveforms of the Q outputs represent the proper binary number after each clock pulse.



▲ **FIGURE 9–58**

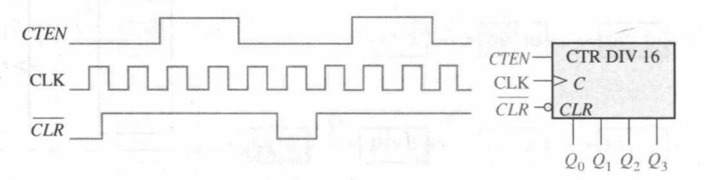
9. By analyzing the J and K inputs to each flip-flop prior to each clock pulse, prove that the decade counter in Figure 9–59 progresses through a BCD sequence. Explain how these conditions in each case cause the counter to go to the next proper state.



▲ **FIGURE 9–59**

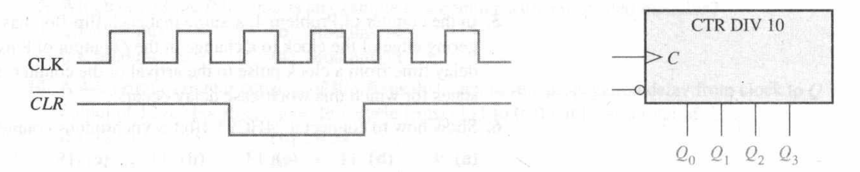
10. The waveforms in Figure 9–60 are applied to the count enable, clear, and clock inputs as indicated. Show the counter output waveforms in proper relation to these inputs. The clear input is asynchronous.

► **FIGURE 9–60**



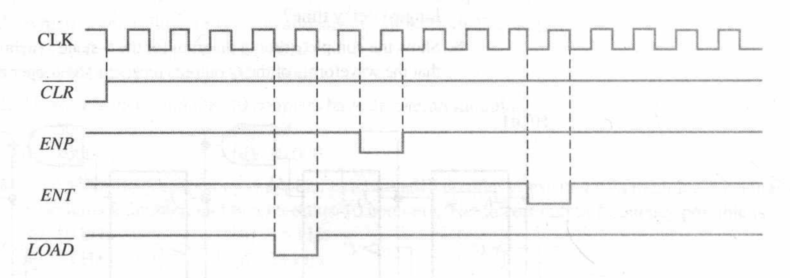
11. A BCD decade counter is shown in Figure 9–61. The waveforms are applied to the clock and clear inputs as indicated. Determine the waveforms for each of the counter outputs (Q_0, Q_1, Q_2, and Q_3). The clear is synchronous, and the counter is initially in the binary 1000 state.

FIGURE 9–61



12. The waveforms in Figure 9–62 are applied to a 74HC163 binary counter. Determine the Q outputs and the RCO. The inputs are $D_0 = 1$, $D_1 = 1$, $D_2 = 0$, and $D_3 = 1$.

FIGURE 9–62



13. The waveforms in Figure 9–62 are applied to a 74HC161 counter. Determine the Q outputs and the RCO. The inputs are $D_0 = 1$, $D_1 = 0$, $D_2 = 0$, and $D_3 = 1$.

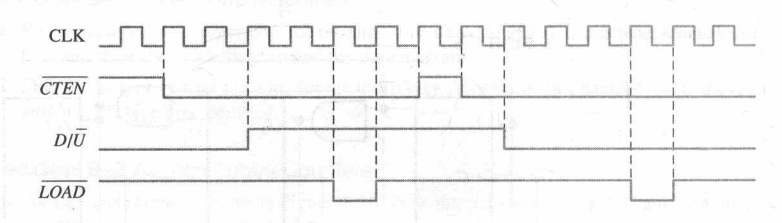
Section 9–4 Up/Down Synchronous Counters

14. Show a complete timing diagram for a 3-bit up/down counter that goes through the following sequence. Indicate when the counter is in the UP mode and when it is in the DOWN mode. Assume positive edge-triggering.

0, 1, 2, 3, 2, 1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1, 0

15. Develop the Q output waveforms for a 74HC190 up/down counter with the input waveforms shown in Figure 9–63. A binary 0 is on the data inputs. Start with a count of 0000.

FIGURE 9–63



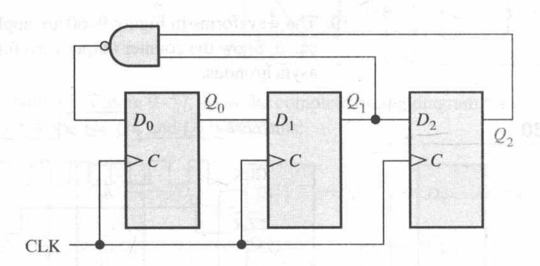
16. Repeat Problem 15 if the $D/\overline{U}$ input signal is inverted with the other inputs the same.

17. Repeat Problem 15 if the $\overline{CTEN}$ is inverted with the other inputs the same.

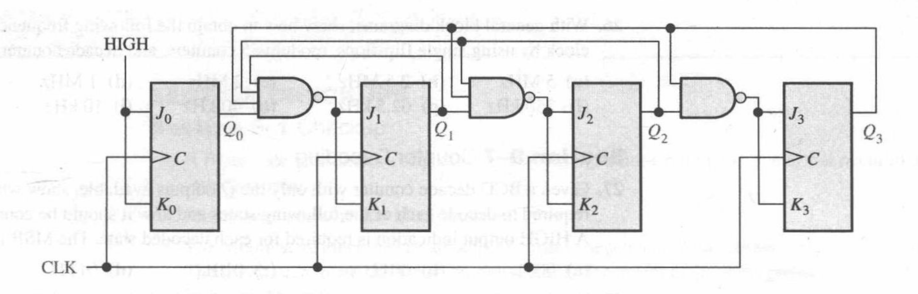
Section 9–5 Design of Synchronous Counters

18. Determine the sequence of the counter in Figure 9–64.

FIGURE 9–64



19. Determine the sequence of the counter in Figure 9–65. Begin with the counter cleared.



▲ FIGURE 9–65

20. Design a counter to produce the following sequence. Use J-K flip-flops.

00, 10, 01, 11, 00, ...

21. Design a counter to produce the following binary sequence. Use J-K flip-flops.

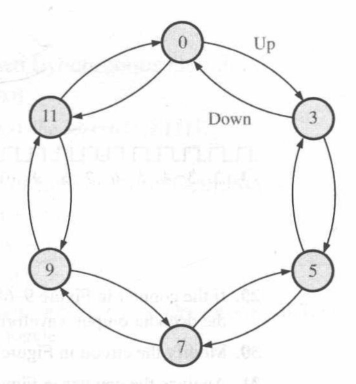
1, 4, 3, 5, 7, 6, 2, 1, ...

22. Design a counter to produce the following binary sequence. Use J-K flip-flops.

0, 9, 1, 8, 2, 7, 3, 6, 4, 5, 0, ...

23. Design a binary counter with the sequence shown in the state diagram of Figure 9–66.

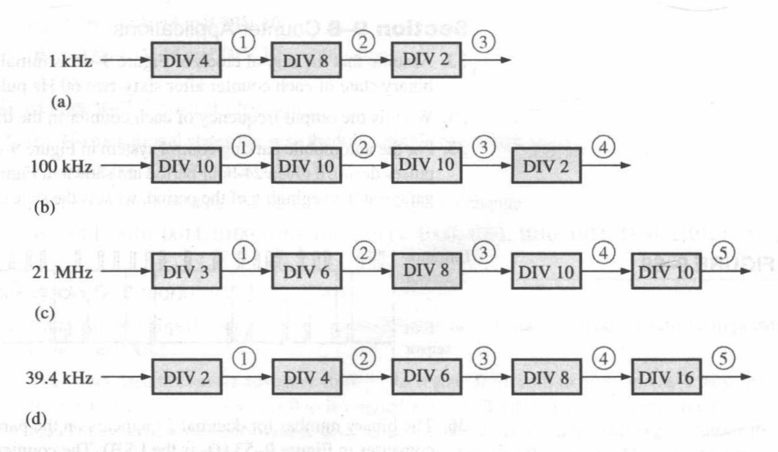
▶ FIGURE 9–66



Section 9–6 Cascaded Counters

24. For each of the cascaded counter configurations in Figure 9–67, determine the frequency of the waveform at each point indicated by a circled number, and determine the overall modulus.

▶ FIGURE 9–67



25. Expand the counter in Figure 9–38 to create a divide-by-10,000 counter and a divide-by-100,000 counter.

26. With general block diagrams, show how to obtain the following frequencies from a 10 MHz clock by using single flip-flops, modulus-5 counters, and decade counters:

(a) 5 MHz **(b)** 2.5 MHz **(c)** 2 MHz **(d)** 1 MHz **(e)** 500 kHz
(f) 250 kHz **(g)** 62.5 kHz **(h)** 40 kHz **(i)** 10 kHz **(j)** 1 kHz

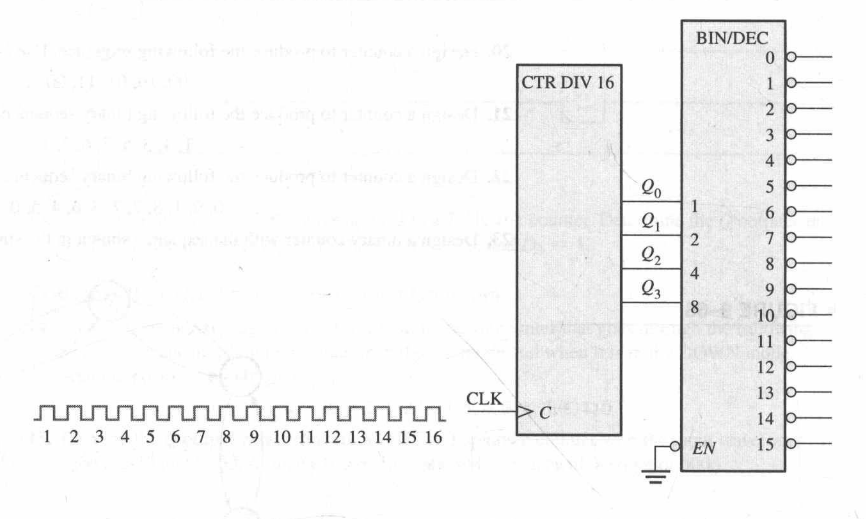
Section 9–7 Counter Decoding

27. Given a BCD decade counter with only the Q outputs available, show what decoding logic is required to decode each of the following states and how it should be connected to the counter. A HIGH output indication is required for each decoded state. The MSB is to the left.

(a) 0001 **(b)** 0011 **(c)** 0101 **(d)** 0111 **(e)** 1000

28. For the 4-bit binary counter connected to the decoder in Figure 9–68, determine each of the decoder output waveforms in relation to the clock pulses.

▶ **FIGURE 9–68**



29. If the counter in Figure 9–68 is asynchronous, determine where the decoding glitches occur on the decoder output waveforms.

30. Modify the circuit in Figure 9–68 to eliminate decoding glitches.

31. Analyze the counter in Figure 9–42 for the occurrence of glitches on the decode gate output. If glitches occur, suggest a way to eliminate them.

32. Analyze the counter in Figure 9–43 for the occurrence of glitches on the outputs of the decoding gates. If glitches occur, make a design change that will eliminate them.

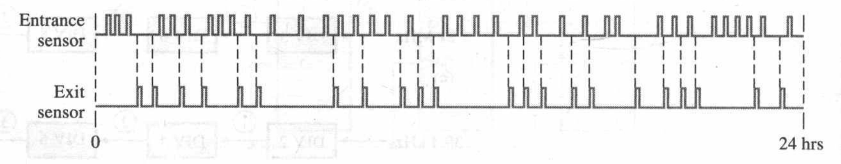
Section 9–8 Counter Applications

33. Assume that the digital clock of Figure 9–48 is initially reset to 12 o'clock. Determine the binary state of each counter after sixty-two 60 Hz pulses have occurred.

34. What is the output frequency of each counter in the digital clock circuit of Figure 9–48?

35. For the automobile parking control system in Figure 9–51, a pattern of entrance and exit sensor pulses during a given 24-hour period are shown in Figure 9–69. If there were 53 cars already in the garage at the beginning of the period, what is the state of the counter at the end of the 24 hours?

▶ **FIGURE 9–69**



36. The binary number for decimal 57 appears on the parallel data inputs of the parallel-to-serial converter in Figure 9–53 (D_0 is the LSB). The counter initially contains all zeros and a 10 kHz clock is applied. Develop the timing diagram showing the clock, the counter outputs, and the serial data output.

ANSWERS

SECTION CHECKUPS

Section 9–1 Checkup

1. A finite state machine is a sequential circuit having a finite number of states that occur in a specified order.
2. Moore state machine and Mealy state machine
3. The Moore state machine has an output(s) that is dependent on the present internal state only. The Mealy state machine has an output(s) that is dependent on both the present internal state and the value of the inputs.

Section 9–2 Asynchronous Counters

1. Asynchronous means that each flip-flop after the first one is enabled by the output of the preceding flip-flop.
2. A modulus-14 counter has fourteen states requiring four flip-flops.

Section 9–3 Synchronous Counters

1. All flip-flops in a synchronous counter are clocked simultaneously.
2. The counter can be preset (initialized) to any given state.
3. Counter is enabled when *ENP* and *ENT* are both HIGH; *RCO* goes HIGH when final state in sequence is reached.

Section 9–4 Up/Down Synchronous Counters

1. The counter goes to 1001.
2. UP: 1111: DOWN: 0000; the next state is 1111.

Section 9–5 Design of Synchronous Counters

1. $J = 1, K =$ X ("don't care")
2. $J = X$ ("don't care"), $K = 0$
3. **(a)** The next state is 1011.

   (b) Q_3 (MSB): no-change or SET; Q_2: no-change or RESET; Q_1: no change or SET; Q_0 (LSB): SET or toggle

Section 9–6 Cascaded Counters

1. Three decade counters produce ÷ 1000; 4 decade counters produce ÷ 10,000.
2. **(a)** ÷ 20: flip-flop and DIV 10

   (b) ÷ 32: flip-flop and DIV 16

   (c) ÷ 160: DIV 16 and DIV 10

   (d) ÷ 320: DIV 16 and DIV 10 and flip-flop

Section 9–7 Counter Decoding

1. **(a)** No transitional states because there is a single bit change

   (b) 0000, 0001, 0010, 0101, 0110, 0111

   (c) No transitional states because there is a single bit change

   (d) 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110

Section 9–8 Counter Applications

1. Gate G_1 resets flip-flop on first clock pulse after count 9. Gate G_2 decodes count 12 to preset counter to 0001.
2. The hours decade counter advances through each state from zero to nine, and as it recycles from nine back to zero, the flip-flop is toggled to the SET state. This produces a ten (10) on the display. When the hours decade counter is in state 12, the decode NAND gate causes the counter to recycle to state 1 on the next clock pulse. The flip-flop resets. This results in a one (01) on the display.

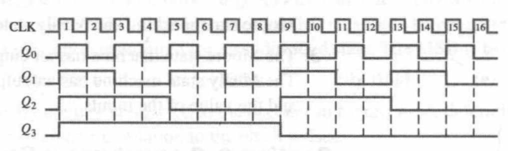
Section 9–9 Logic Symbols with Dependency Notation

1. *C:* control, usually clock; *M:* mode; *G:* AND
2. *D* indicates data storage.

RELATED PROBLEMS FOR EXAMPLES

9–1 See Figure 9–70.

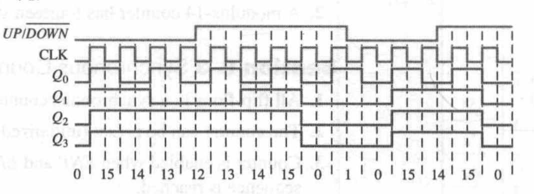
▶ **FIGURE 9–70**



9–2 Connect Q_0 to the NAND gate as a third input (Q_2 and Q_3 are two of the inputs). Connect the $\overline{CLR}$ line to the $\overline{CLR}$ input of FF0 as well as FF2 and FF3.

9–3 See Figure 9–71.

▶ **FIGURE 9–71**



9–4 See Table 9–14.

▶ **TABLE 9–14**

Present Invalid State			D Inputs			Next State			
Q_2	Q_1	Q_0	D_2	D_1	D_0	Q_2	Q_1	Q_0	
0	0	0	1	1	1	1	1	1	valid state
0	1	1	0	0	0	0	0	0	
1	0	0	1	1	1	1	1	1	
1	1	0	1	0	1	1	0	1	valid state

000 → 111
011 → 000 → 111
100 → 111
110 → 101

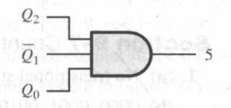
9–5 Three flip-flops, sixteen 3-input AND gates, two 4-input OR gates, four 2-input OR gates, and one inverter

9–6 Five decade counters are required. $10^5 = 100{,}000$

9–7 $f_{Q0} = 1\text{ MHz}/[(10)(2)] = 50\text{ kHz}$

9–8 See Figure 9–72.

▶ **FIGURE 9–72**



TRUE/FALSE QUIZ

1. F **2.** T **3.** T **4.** F **5.** T **6.** F **7.** T **8.** F **9.** T **10.** F

SELF-TEST

1. (c) **2.** (d) **3.** (a) **4.** (b) **5.** (b) **6.** (c) **7.** (a) **8.** (c)
9. (b) **10.** (c) **11.** (d) **12.** (a) **13.** (c) **14.** (b) **15.** (b) **16.** (d)

Chapter 10 Data Storage

CHAPTER OUTLINE

10-1 Semiconductor Memory Basics

InfoNote

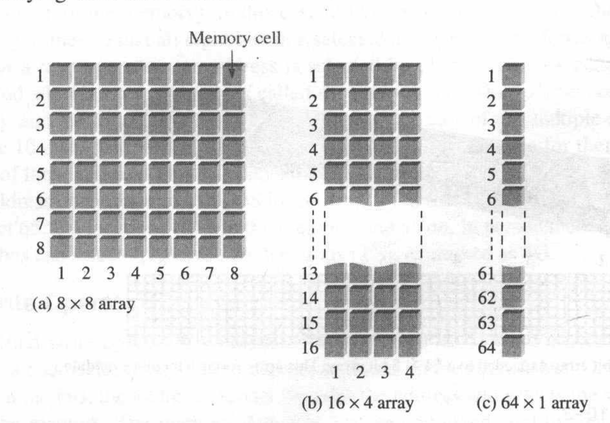
The general definition of *word* is a complete unit of information consisting of a unit of binary data. When applied to computer instructions, a word is more specifically defined as two bytes (16 bits). As an important part of assembly language used in computers, the DW (Define Word) directive means to define data in 16-bit units. This definition is independent of the particular microprocessor or the size of its data bus. Assembly language also allows definitions of bytes (8 bits) with the DB directive, double words (32 bits) with the DD directive, and quad-words (64 bits) with the QD directive.

Units of Binary Data: Bits, Bytes, Nibbles, and Words

As a rule, memories store data in units that have from one to eight bits. The smallest unit of binary data, as you know, is the **bit**. In many applications, data are handled in an 8-bit unit called a **byte** or in multiples of 8-bit units. The byte can be split into two 4-bit units that are called **nibbles**. Bytes can also be grouped into words. The term **word** can have two meanings in computer terminology. In memories, it is defined as a group of bits or bytes that acts as a single entity that can be stored in one memory location. In assembly language, a word is specifically defined as two bytes.

The Basic Memory Array

Each storage element in a memory can retain either a 1 or a 0 and is called a **cell**. Memories are made up of arrays of cells, as illustrated in Figure 10–1 using 64 cells as an example. Each block in the **memory array** represents one storage cell, and its location can be identified by specifying a row and a column.



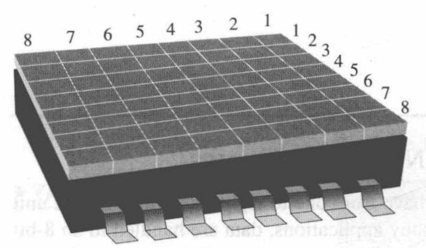
▲ FIGURE 10–1

A 64-cell memory array organized in three different ways.

The 64-cell array can be organized in several ways based on units of data. Figure 10–1(a) shows an 8 × 8 array, which can be viewed as either a 64-bit memory or an 8-byte memory. Part (b) shows a 16 × 4 array, which is a 16-nibble memory, and part (c) shows a 64 × 1 array, which is a 64-bit memory. A memory is identified by the number of words it can store times the word size. For example, a 16k × 8 memory can store 16,384 words of eight bits each. The inconsistency here is common in memory terminology. The actual number of words is always a power of 2, which, in this case, is $2^{14} = 16{,}384$. However, it is common practice to state the number to the nearest thousand, in this case, 16k.

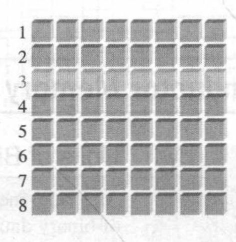
Memory Address and Capacity

A representation of a small 8 × 8 memory chip is shown in Figure 10–2(a). The location of a unit of data in a memory array is called its **address**. For example, in part (b), the address of a bit in the 2-dimensional array is specified by the row and column as shown. In part (c), the address of a byte is specified only by the row. So, as you can see, the address depends on how the memory is organized into units of data. Personal computers have random-access memories organized in bytes. This means that the smallest group of bits that can be addressed is eight.



(a) Physical structure of 64-bit memory.

(b) The address of the blue bit is row 5, column 4.

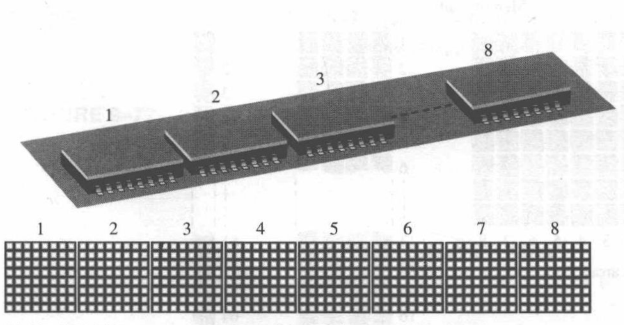


(c) The address of the blue byte is row 3.

▲ **FIGURE 10–2**

Examples of memory address in a 2-dimensional memory array.

Figure 10–3(a) illustrates the expansion of the 8 × 8 (64-bit) array to a 64-byte memory. The address of a byte in the array is specified by the row and column, as shown. In this case, the smallest group of bits that can be accessed is eight. This can be viewed as a 3-dimensional array, as shown in part (b).



(a) The 8 × 8 bit array expanded to a 64 × 8 bit array. This array forms a memory module.

(b) The address of the blue byte is row 5, column 8.

▲ **FIGURE 10–3**

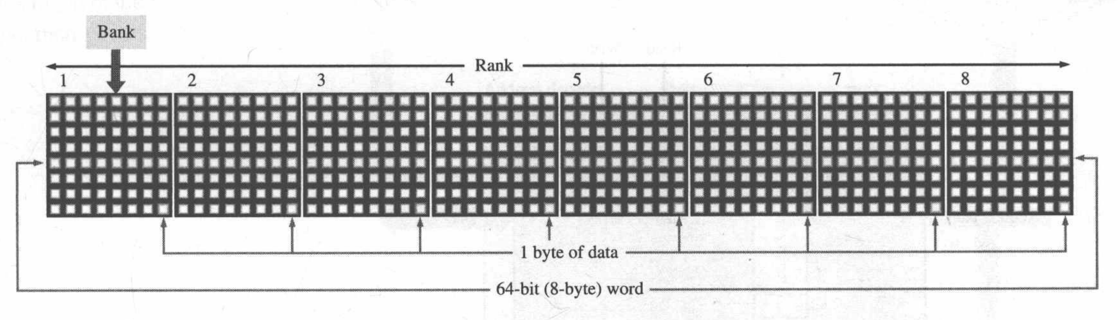
Example of memory address in an expanded (multiple) array.

The **capacity** of a memory is the total number of data units that can be stored. For example, in the bit-organized memory array in Figure 10–2(b), the capacity is 64 bits. In the byte-organized memory array in Figure 10–2(c), the capacity is 8 bytes, which is also

64 bits. In Figure 10–3, the capacity is 64 bytes. Computer memories typically have multiple gigabytes of internal memory. Computers usually transfer and store data as 64-bit words, in which case all eight bits of row five in each chip in Figure 10–3(a) would be accessed.

Memory Banks and Ranks

A **bank** is a section of memory within a single memory array (chip). A memory chip may have one or more banks. Memory banks can be used for storing frequently used information. Easier and faster access can be achieved by knowing the section of memory in which the data are stored. A **rank** is a group of chips that make up a memory module that stores data in units such as words or bytes. These terms are illustrated in Figure 10–4.



▲ **FIGURE 10–4**

Simple illustration of memory bank and memory rank.

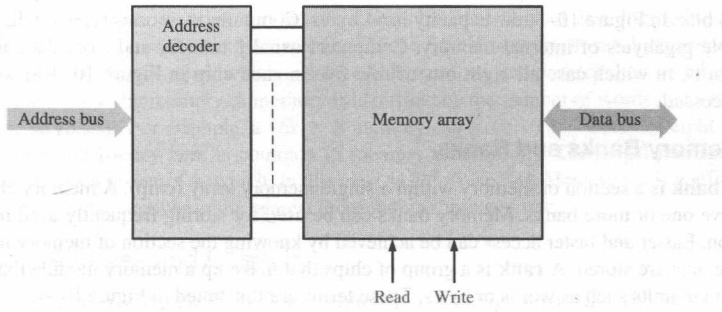
Basic Memory Operations

Addressing is the process of accessing a specified location in memory. Since a memory stores binary data, data must be put into the memory and data must be copied from the memory when needed. The **write** operation puts data into a specified address in the memory, and the **read** operation copies data out of a specified address in the memory. The addressing operation, which is part of both the write and the read operations, selects the specified memory address.

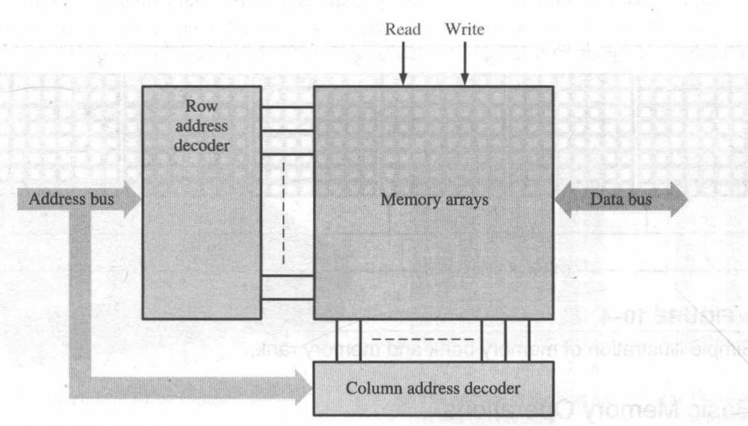
Data units go into the memory during a write operation and come out of the memory during a read operation on a set of lines called the *data bus.* As indicated in Figure 10–5, the data bus is bidirectional, which means that data can go in either direction (into the memory or out of the memory). In this case of byte-organized memories, the data bus has at least eight lines so that all eight bits in a selected address are transferred in parallel. For a write or a read operation, an address is selected by placing a binary code representing the desired address on a set of lines called the *address bus.* The address code is decoded internally, and the appropriate address is selected. In the case of the multiple-array memory in Figure 10–5(b) there are two decoders, one for the rows and one for the columns. The number of lines in the address bus depends on the capacity of the memory. For example, a 15-bit address code can select 32,768 locations (2^{15}) in the memory, a 16-bit address code can select 65,536 locations (2^{16}) in the memory, and so on. In personal computers a 32-bit address bus can select 4,294,967,296 locations (2^{32}), expressed as 4G.

The Write Operation

A simplified write operation is illustrated in Figure 10–6. To store a byte of data in the memory, a code held in the address register is placed on the address bus. Once the address code is on the bus, the address decoder decodes the address and selects the specified location in the memory. The memory then gets a write command, and the data byte held in the data register is placed on the data bus and stored in the selected memory address, thus completing the write operation. When a new data byte is written into a memory address, the current data byte stored at that address is overwritten (replaced with a new data byte).

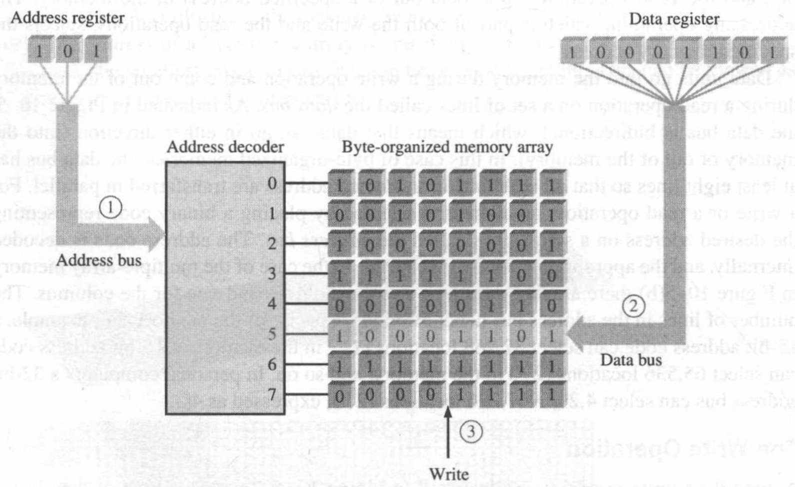


(a) Single-array memory



(b) Multiple-array memory

◄FIGURE 10–5
Block diagram of a single-array memory and a multiple-array memory showing address bus, address decoder(s), bidirectional data bus, and read/write inputs.



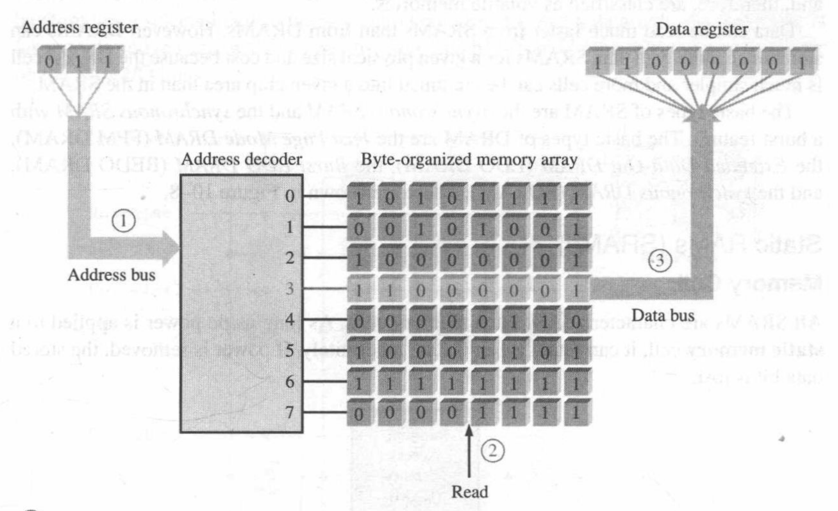
① Address code 101 is placed on the address bus and address 5 is selected.
② Data byte is placed on the data bus.
③ Write command causes the data byte to be stored in address 5, replacing previous data.

◄FIGURE 10–6
Illustration of the write operation.

The Read Operation

A simplified read operation is illustrated in Figure 10–7. Again, a code held in the address register is placed on the address bus. Once the address code is on the bus, the address decoder decodes the address and selects the specified location in the memory. The memory then gets a read command, and a "copy" of the data byte that is stored in the selected memory address is placed on the data bus and loaded into the data register, thus completing the read operation. When a data byte is read from a memory address, it also remains stored at that address. This is called *nondestructive read.*

▶ **FIGURE 10–7**
Illustration of the read operation.



① Address code 011 is placed on the address bus and address 3 is selected.
② Read command is applied.
③ The contents of address 3 is placed on the data bus and shifted into data register. The contents of address 3 is not erased by the read operation.

RAMs and ROMs

The two major categories of semiconductor memories are the RAM and the ROM. **RAM** (random-access memory) is a type of memory in which all addresses are accessible in an equal amount of time and can be selected in any order for a read or write operation. All RAMs have both *read* and *write* capability. Because RAMs lose stored data when the power is turned off, they are **volatile** memories.

ROM (read-only memory) is a type of memory in which data are stored permanently or semipermanently. Data can be read from a ROM, but there is no write operation as in the RAM. The ROM, like the RAM, is a random-access memory but the term *RAM* traditionally means a random-access *read/write* memory. Several types of RAMs and ROMs will be covered in this chapter. Because ROMs retain stored data even if power is turned off, they are **nonvolatile** memories.

SECTION 10-1 CHECKUP

Answers are at the end of the chapter.

1. What is the smallest unit of data that can be stored in a memory?
2. What is the bit capacity of a memory that can store 256 bytes of data?
3. What is a write operation?
4. What is a read operation?
5. How is a given unit of data located in a memory?
6. Describe the difference between a RAM and a ROM.

10-2 The Random-Access Memory (RAM)

The RAM Family

The two major categories of RAM are the *static RAM* (SRAM) and the *dynamic RAM* (DRAM). **SRAMs** generally use latches as storage elements and can therefore store data indefinitely *as long as dc power is applied.* **DRAMs** use capacitors as storage elements and cannot retain data very long without the capacitors being recharged by a process called **refreshing.** Both SRAMs and DRAMs will lose stored data when dc power is removed and, therefore, are classified as volatile memories.

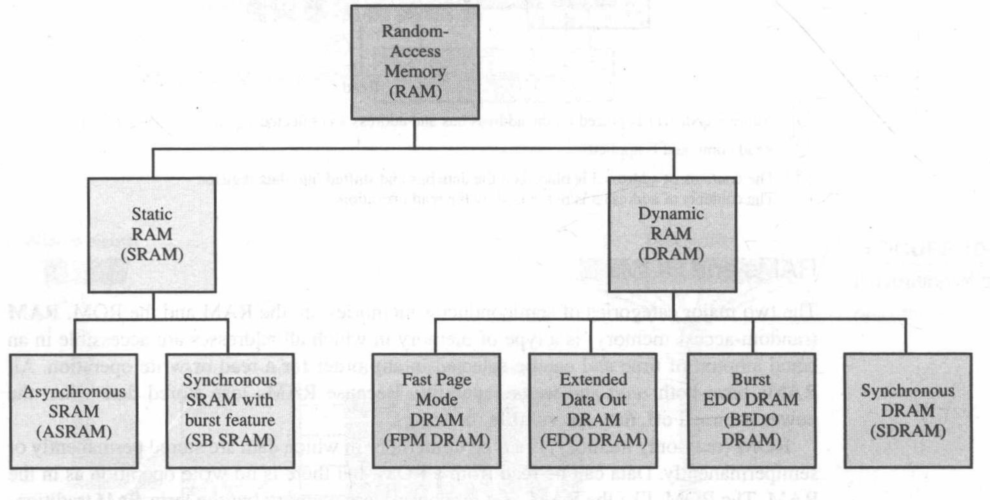
Data can be read much faster from SRAMs than from DRAMs. However, DRAMs can store much more data than SRAMs for a given physical size and cost because the DRAM cell is much simpler and more cells can be crammed into a given chip area than in the SRAM.

The basic types of SRAM are the *asynchronous SRAM* and the *synchronous SRAM* with a burst feature. The basic types of DRAM are the *Fast Page Mode DRAM* (FPM DRAM), the *Extended Data Out DRAM* (EDO DRAM), the *Burst EDO DRAM* (BEDO DRAM), and the *synchronous DRAM* (SDRAM). These are shown in Figure 10–8.

Static RAMs (SRAMs)

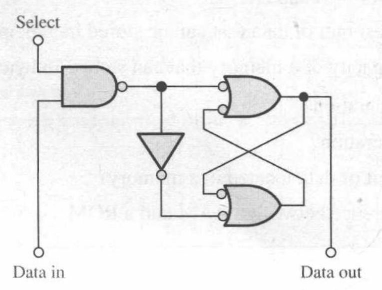
Memory Cell

All SRAMs are characterized by latch memory cells. As long as dc power is applied to a **static memory** cell, it can retain a 1 or 0 state indefinitely. If power is removed, the stored data bit is lost.



▲FIGURE 10–8
The RAM family.

Figure 10–9 shows a basic SRAM latch memory cell. The cell is selected by an active level on the Select line and a data bit (1 or 0) is written into the cell by placing it on the Data in line. A data bit is read by taking it off the Data out line.

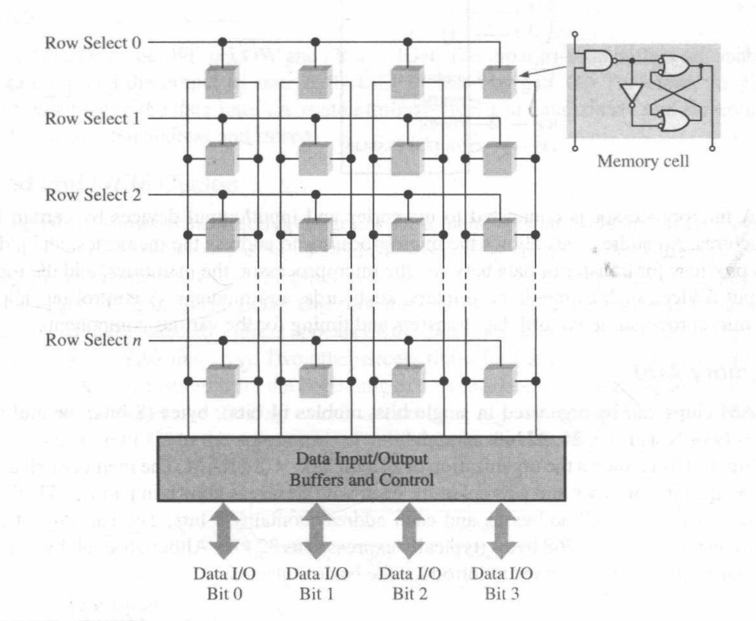


◀FIGURE 10–9
A typical SRAM latch memory cell.

Static Memory Cell Array

The memory cells in a SRAM are organized in rows and columns, as illustrated in Figure 10–10 for the case of an $n \times 4$ array. All the cells in a row share the same Row Select line. Each set of Data in and Data out lines go to each cell in a given column and are connected to a single data line that serves as both an input and output (Data I/O) through the data input and data output buffers.

To write a data unit, in this case a nibble (4 bits), into a given row of cells in the memory array, the Row Select line is taken to its active state and four data bits are placed on the Data I/O lines. The Write line is then taken to its active state, which causes each data bit to be stored in a selected cell in the associated column. To read a data unit, the Read line is taken to its active state, which causes the four data bits stored in the selected row to appear on the Data I/O lines.



▶ **FIGURE 10–10**
Basic SRAM array.

Basic Asynchronous SRAM Organization

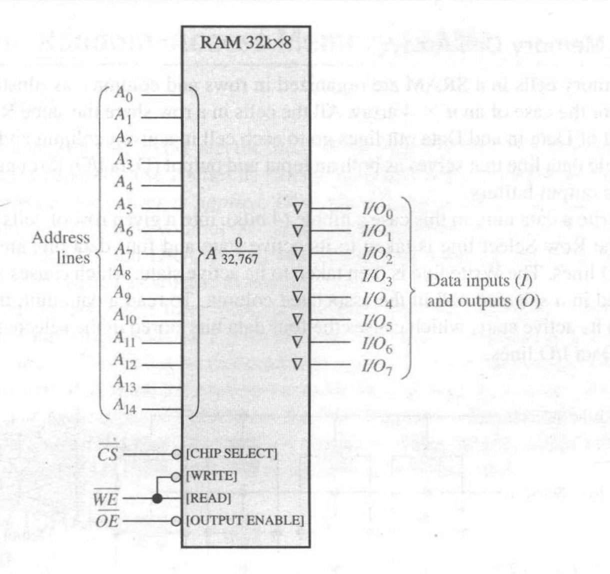
An asynchronous SRAM is one in which the operation is not synchronized with a system clock. To illustrate the general organization of a SRAM, a 32k × 8 bit memory is used. A logic symbol for this memory is shown in Figure 10–11.

In the READ mode, the eight data bits that are stored in a selected address appear on the data output lines. In the WRITE mode, the eight data bits that are applied to the data input lines are stored at a selected address. The data input and data output lines (I/O_0 through I/O_7) share the same lines. During READ, they act as output lines (O_0 through O_7) and during WRITE they act as input lines (I_0 through I_7).

Tri-state Outputs and Buses

Tri-state buffers in a memory allow the data lines to act as either input or output lines and connect the memory to the data bus in a computer. These buffers have three output states: HIGH (1), LOW (0), and HIGH-Z (open). Tri-state outputs are indicated on logic symbols by a small inverted triangle (∇), as shown in Figure 10–11, and are used for compatibility with bus structures such as those found in microprocessor-based systems.

Physically, a **bus** is one or more conductive paths that serve to interconnect two or more functional components of a system or several diverse systems. Electrically, a bus is a collection of specified voltage levels and/or current levels and signals that allow various devices to communicate and work properly together.



◄ **FIGURE 10–11**

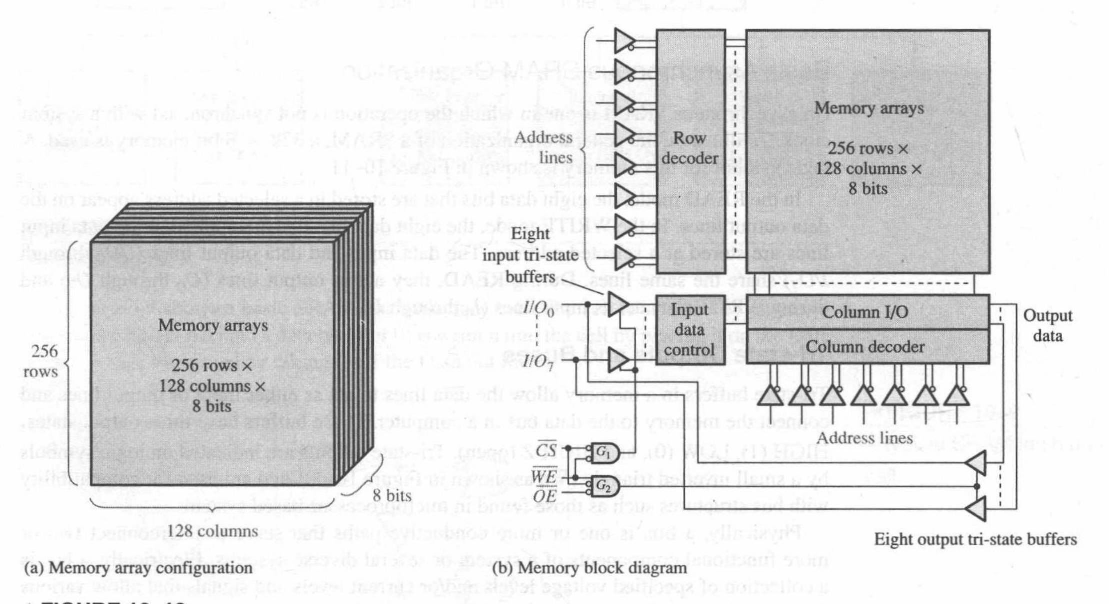
Logic diagram for an asynchronous 32k × 8 SRAM.

A microprocessor is connected to memories and input/output devices by certain bus structures. An address bus allows the microprocessor to address the memories, and a data bus provides for transfer of data between the microprocessor, the memories, and the input/output devices such as monitors, printers, keyboards, and modems. A control bus allows the microprocessor to control data transfers and timing for the various components.

Memory Array

SRAM chips can be organized in single bits, nibbles (4 bits), bytes (8 bits), or multiple bytes (words with 16, 24, 32 bits, etc.).

Figure 10–12 shows the organization of a small 32k × 8 SRAM. The memory cell array is arranged in 256 rows and 128 columns, each with 8 bits, as shown in part (a). There are actually $2^{15} = 32,768$ addresses and each address contains 8 bits. The capacity of this example memory is 32,768 bytes (typically expressed as 32 kB). Although small by today's standards, this memory serves to introduce the basic concepts.



(a) Memory array configuration (b) Memory block diagram

▲ **FIGURE 10–12**

Basic organization of an asynchronous 32k × 8 SRAM.

The SRAM in Figure 10–12(b) works as follows. First, the chip select, $\overline{CS}$, must be LOW for the memory to operate. (Other terms for chip select are *enable* or *chip enable.*) Eight of the fifteen address lines are decoded by the row decoder to select one of the 256 rows. Seven of the fifteen address lines are decoded by the column decoder to select one of the 128 8-bit columns.

Read

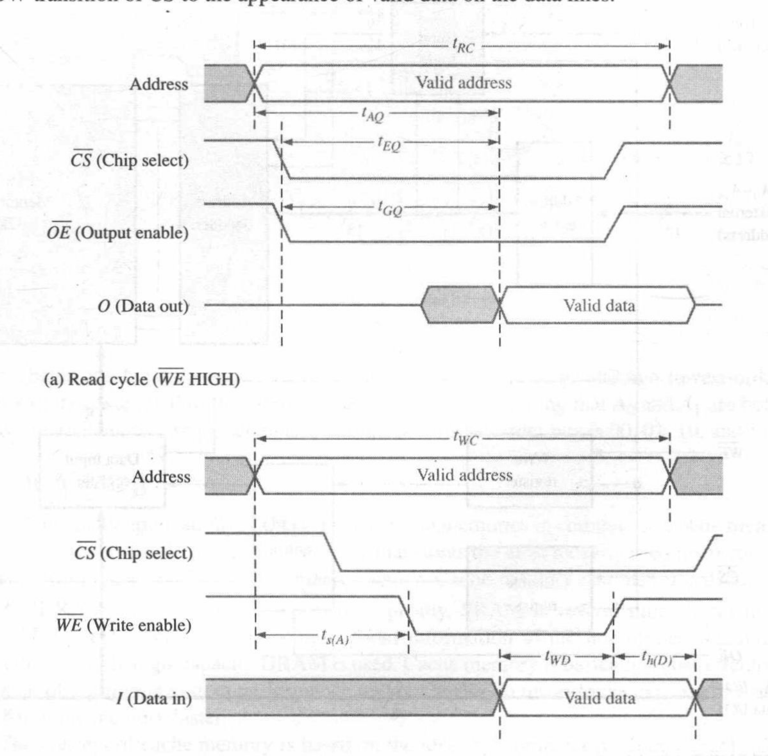
In the READ mode, the write enable input, $\overline{WE}$, is HIGH and the output enable, $\overline{OE}$, is LOW. The input tri-state buffers are disabled by gate G_1, and the column output tri-state buffers are enabled by gate G_2. Therefore, the eight data bits from the selected address are routed through the column I/O to the data lines (I/O_0 though I/O_7), which are acting as data output lines.

Write

In the WRITE mode, $\overline{WE}$ is LOW and $\overline{OE}$ is HIGH. The input tri-state buffers are enabled by gate G_1, and the output tri-state buffers are disabled by gate G_2. Therefore, the eight input data bits on the data lines are routed through the input data control and the column I/O to the selected address and stored.

Read and Write Cycles

Figure 10–13 shows typical timing diagrams for a memory read cycle and a write cycle. For the read cycle shown in part (a), a valid address code is applied to the address lines for a specified time interval called the *read cycle time,* t_{RC}. Next, the chip select ($\overline{CS}$) and the output enable ($\overline{OE}$) inputs go LOW. One time interval after the $\overline{OE}$ input goes LOW, a valid data byte from the selected address appears on the data lines. This time interval is called the *output enable access time,* t_{GQ}. Two other access times for the read cycle are the *address access time,* t_{AQ}, measured from the beginning of a valid address to the appearance of valid data on the data lines and the *chip enable access time,* t_{EQ}, measured from the HIGH-to-LOW transition of $\overline{CS}$ to the appearance of valid data on the data lines.



▶ FIGURE 10–13
Timing diagrams for typical read and write cycles for the SRAM in Figure 10–12.

During each read cycle, one unit of data, a byte in this case, is read from the memory.

For the write cycle shown in Figure 10–13(b), a valid address code is applied to the address lines for a specified time interval called the *write cycle time,* t_{WC}. Next, the chip select ($\overline{CS}$) and the write enable ($\overline{WE}$) inputs go LOW. The required time interval from the beginning of a valid address until the $\overline{WE}$ input goes LOW is called the *address setup time,* $t_{s(A)}$. The time that the $\overline{WE}$ input must be LOW is the write pulse width. The time that the input $\overline{WE}$ must remain LOW after valid data are applied to the data inputs is designated t_{WD}; the time that the valid input data must remain on the data lines after the $\overline{WE}$ input goes HIGH is the *data hold time,* $t_{h(D)}$.

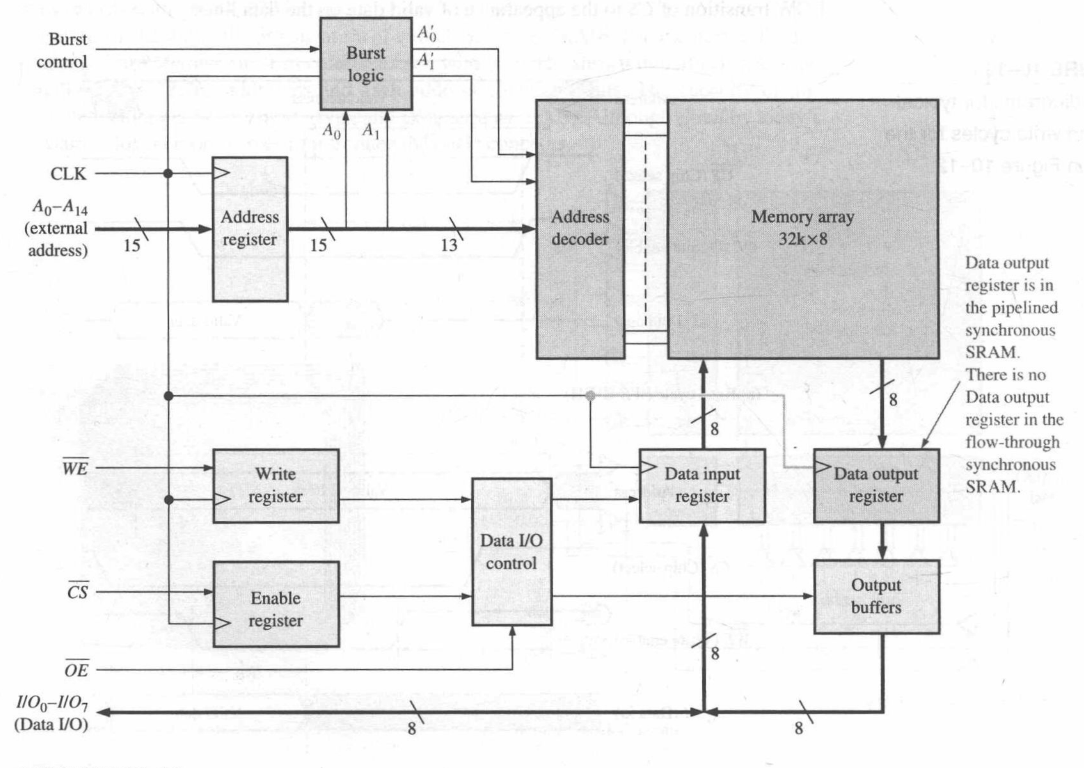
During each write cycle, one unit of data is written into the memory.

Synchronous SRAM with Burst Feature

Unlike the asynchronous SRAM, a synchronous SRAM is synchronized with the system clock. For example, in a computer system, the synchronous SRAM operates with the same clock signal that operates the microprocessor so that the microprocessor and memory are synchronized for faster operation.

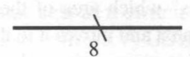
The fundamental concept of the synchronous feature of a SRAM can be shown with Figure 10–14, which is a simplified block diagram of a 32k × 8 memory for purposes of illustration. The synchronous SRAM is similar to the asynchronous SRAM in terms of the memory array, address decoder, and read/write and enable inputs. The basic difference is that the synchronous SRAM uses clocked registers to synchronize all inputs with the system clock. The address, the read/write input, the chip enable, and the input data are all latched into their respective registers on an active clock pulse edge. Once this information is latched, the memory operation is in sync with the clock.

For the purpose of simplification, a notation for multiple parallel lines or bus lines is introduced in Figure 10–14, as an alternative to drawing each line separately. A set of parallel lines can be indicated by a single heavy line with a slash and the number of separate lines in the set. For example, the following notation represents a set of 8 parallel lines:



▲ **FIGURE 10–14**

A basic block diagram of a synchronous SRAM with burst feature.



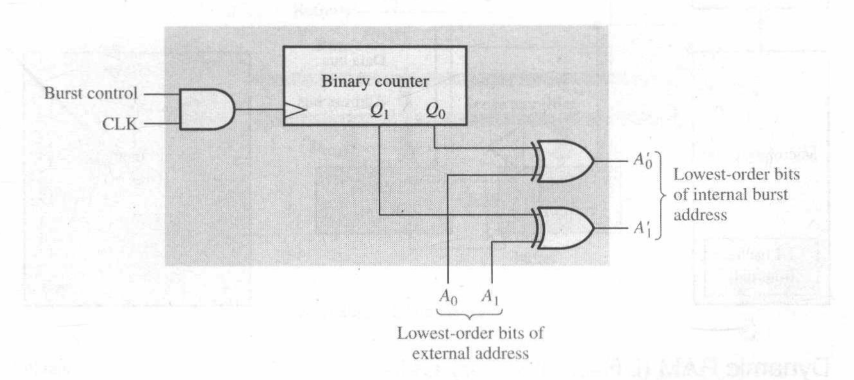
The address bits A_0 through A_{14} are latched into the Address register on the positive edge of a clock pulse. On the same clock pulse, the state of the write enable ($\overline{WE}$) line and chip select ($\overline{CS}$) are latched into the Write register and the Enable register respectively. These are one-bit registers or simply flip-flops. Also, on the same clock pulse the input data are latched into the Data input register for a Write operation, and data in a selected memory address are latched into the Data output register for a Read operation, as determined by the Data I/O control based on inputs from the Write register, Enable register, and the Output enable ($\overline{OE}$).

Two basic types of synchronous SRAM are the *flow-through* and the *pipelined.* The flow-through synchronous SRAM does not have a Data output register, so the output data flow asynchronously to the data I/O lines through the output buffers. The **pipelined** synchronous SRAM has a Data output register, as shown in Figure 10–14, so the output data are synchronously placed on the data I/O lines.

The Burst Feature

As shown in Figure 10–14, synchronous SRAMs normally have an address burst feature, which allows the memory to read or write up to four sequential locations using a single address. When an external address is latched in the address register, the two lowest-order address bits, A_0 and A_1, are applied to the burst logic. This produces a sequence of four internal addresses by adding 00, 01, 10, and 11 to the two lowest-order address bits on successive clock pulses. The sequence always begins with the base address, which is the external address held in the address register.

The address burst logic in a typical synchronous SRAM consists of a binary counter and exclusive-OR gates, as shown in Figure 10–15. For 2-bit burst logic, the internal burst address sequence is formed by the base address bits A_2–A_{14} plus the two burst address bits A'_1 and A'_0.



▶ FIGURE 10–15
Address burst logic.

To begin the burst sequence, the counter is in its 00 state and the two lowest-order address bits are applied to the inputs of the XOR gates. Assuming that A_0 and A_1 are both 0, the internal address sequence in terms of its two lowest-order bits is 00, 01, 10, and 11.

Cache Memory

One of the major applications of SRAMs is in cache memories in computers. **Cache memory** is a relatively small, high-speed memory that stores the most recently used instructions or data from the larger but slower main memory. Cache memory can also use dynamic RAM (DRAM), which is discussed next. Typically, SRAM is several times faster than DRAM. Overall, a cache memory gets stored information to the microprocessor much faster than if only high-capacity DRAM is used. Cache memory is basically a cost-effective method of improving system performance without having to resort to the expense of making all of the memory faster.

The concept of cache memory is based on the idea that computer programs tend to get instructions or data from one area of main memory before moving to another area. Basi-

cally, the cache controller "guesses" which area of the slow dynamic memory the CPU (central-processing unit) will need next and moves it to the cache memory so that it is ready when needed. If the cache controller guesses right, the data are immediately available to the microprocessor. If the cache controller guesses wrong, the CPU must go to the main memory and wait much longer for the correct instructions or data. Fortunately, the cache controller is right most of the time.

Cache Analogy

There are many analogies that can be used to describe a cache memory, but comparing it to a home refrigerator is perhaps the most effective. A home refrigerator can be thought of as a "cache" for certain food items while the supermarket is the main memory where all foods are kept. Each time you want something to eat or drink, you can go to the refrigerator (cache) first to see if the item you want is there. If it is, you save a lot of time. If it is not there, then you have to spend extra time to get it from the supermarket (main memory).

L1 and L2 Caches

A first-level cache (L1 cache) is usually integrated into the processor chip and has a very limited storage capacity. L1 cache is also known as *primary cache*. A second-level cache (L2 cache) may also be integrated into the processor or as a separate memory chip or set of chips external to the processor; it usually has a larger storage capacity than an L1 cache. L2 cache is also known as *secondary cache*. Some systems may have higher-level caches (L3, L4, etc.), but L1 and L2 are the most common. Also, some systems use a disk cache to enhance the performance of the hard disk because DRAM, although much slower than SRAM, is much faster than the hard disk drive. Figure 10–16 illustrates L1 and L2 cache memories in a computer system.

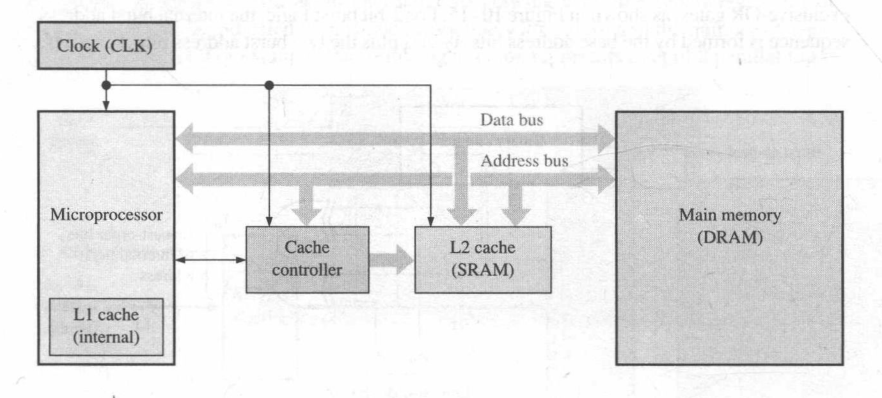


FIGURE 10–16 Block diagram showing L1 and L2 cache memories in a computer system.

Dynamic RAM (DRAM) Memory Cells

Dynamic memory cells store a data bit in a small capacitor rather than in a latch. The advantage of this type of cell is that it is very simple, thus allowing very large memory arrays to be constructed on a chip at a lower cost per bit. The disadvantage is that the storage capacitor cannot hold its charge over an extended period of time and will lose the stored data bit unless its charge is refreshed periodically. To refresh requires additional memory circuitry and complicates the operation of the DRAM. Figure 10–17 shows a typical DRAM cell consisting of a single MOS transistor (MOSFET) and a capacitor.

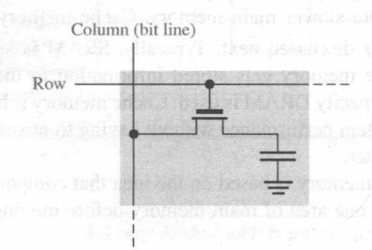


FIGURE 10–17 A MOS DRAM cell.

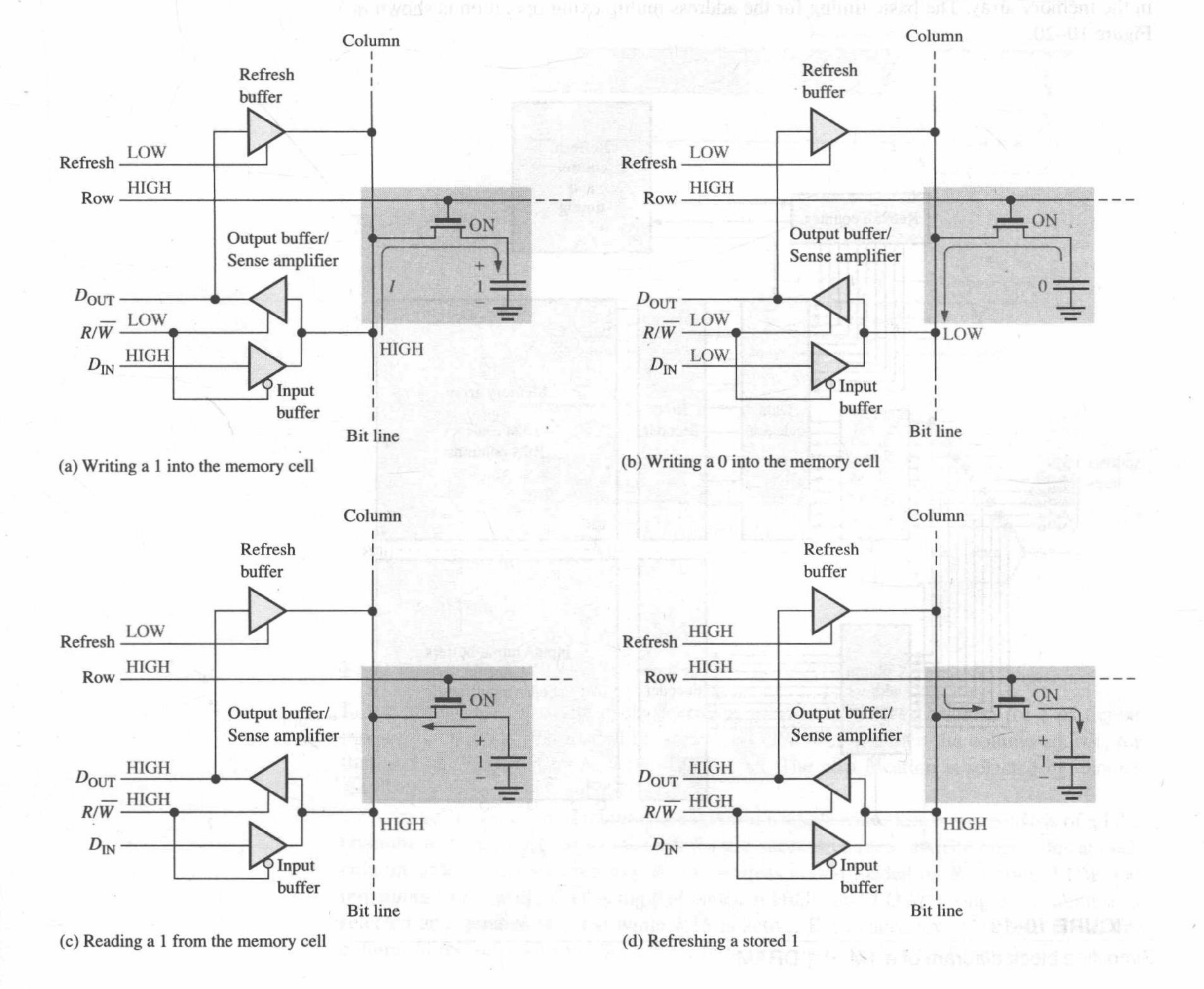
In this type of cell, the transistor acts as a switch. The basic simplified operation is illustrated in Figure 10–18 and is as follows. A LOW on the $R/\overline{W}$ line (WRITE mode) enables the tri-state input buffer and disables the output buffer. For a 1 to be written into the cell, the D_{IN} line must be HIGH, and the transistor must be turned on by a HIGH on the row line. The transistor acts as a closed switch connecting the capacitor to the bit line. This connection allows the capacitor to charge to a positive voltage, as shown in Figure 10–18(a). When a 0 is to be stored, a LOW is applied to the D_{IN} line. If the capacitor is storing a 0, it remains uncharged, or if it is storing a 1, it discharges as indicated in Figure 10–18(b). When the row line is taken back LOW, the transistor turns off and disconnects the capacitor from the bit line, thus "trapping" the charge (1 or 0) on the capacitor.

To read from the cell, the $R/\overline{W}$ (Read/$\overline{\text{Write}}$) line is HIGH, enabling the output buffer and disabling the input buffer. When the row line is taken HIGH, the transistor turns on and connects the capacitor to the bit line and thus to the output buffer (sense amplifier), so the data bit appears on the data-output line (D_{OUT}). This process is illustrated in Figure 10–18(c).

For refreshing the memory cell, the $R/\overline{W}$ line is HIGH, the row line is HIGH, and the refresh line is HIGH. The transistor turns on, connecting the capacitor to the bit line. The output buffer is enabled, and the stored data bit is applied to the input of the refresh buffer, which is enabled by the HIGH on the refresh input. This produces a voltage on the bit line corresponding to the stored bit, thus replenishing the capacitor. This is illustrated in Figure 10–18(d).

▼FIGURE 10–18

Basic operation of a DRAM cell.



(a) Writing a 1 into the memory cell

(b) Writing a 0 into the memory cell

(c) Reading a 1 from the memory cell

(d) Refreshing a stored 1

DRAM Organization

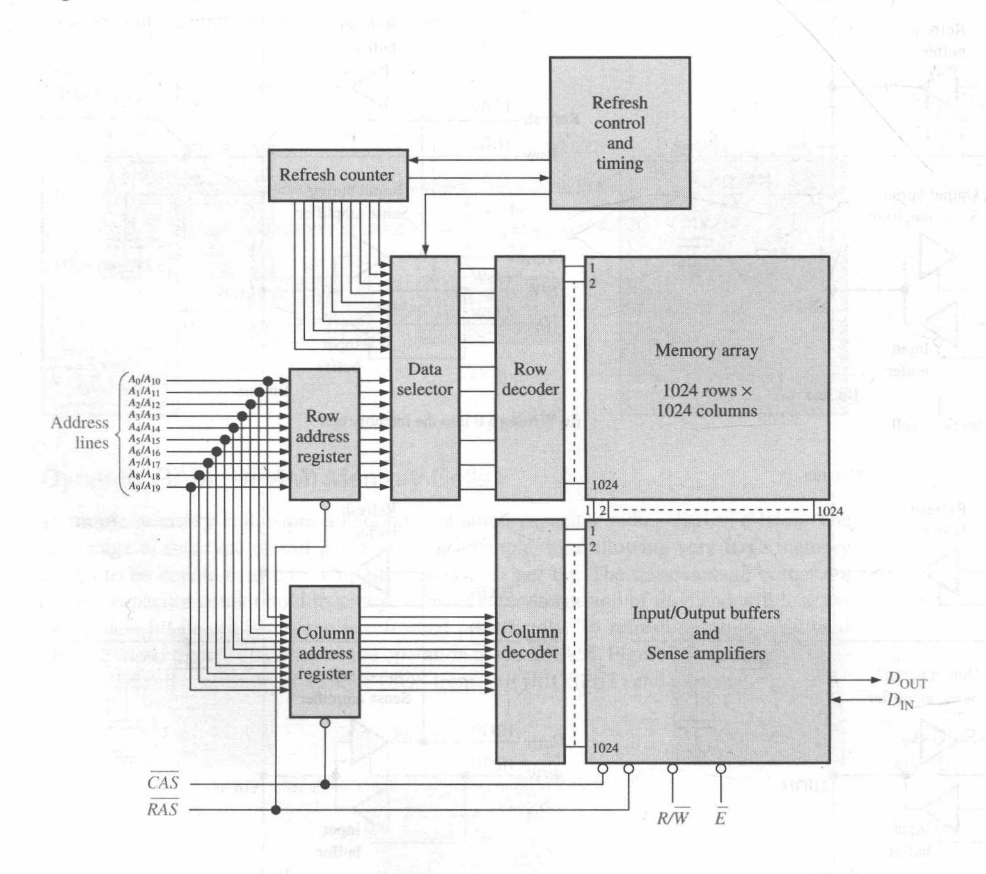
The major application of DRAMs is in the main memory of computers. The difference between DRAMs and SRAMs is the type of memory cell. As you have seen, the DRAM memory cell consists of one transistor and a capacitor and is much simpler than the SRAM cell. This allows much greater densities in DRAMs and results in greater bit capacities for a given chip area, although much slower access time.

Again, because charge stored in a capacitor will leak off, the DRAM cell requires a frequent refresh operation to preserve the stored data bit. This requirement results in more complex circuitry than in a SRAM. Several features common to most DRAMs are now discussed, using a generic 1M × 1 bit DRAM as an example.

Address Multiplexing

DRAMs use a technique called *address multiplexing* to reduce the number of address lines. Figure 10–19 shows the block diagram of a 1,048,576-bit (1 Mb) DRAM with a 1M × 1 organization. We will focus on the blue blocks to illustrate address multiplexing. The green blocks represent the refresh logic.

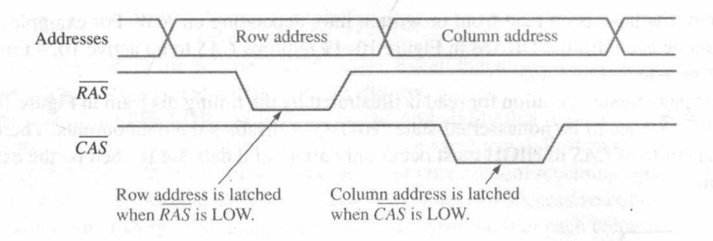
The ten address lines are time multiplexed at the beginning of a memory cycle by the row address select ($\overline{RAS}$) and the column address select ($\overline{CAS}$) into two separate 10-bit address fields. First, the 10-bit row address is latched into the row address register. Next, the 10-bit column address is latched into the column address register. The row address and the column address are decoded to select one of the 1,048,576 addresses ($2^{20} = 1,048,576$) in the memory array. The basic timing for the address multiplexing operation is shown in Figure 10–20.



▲ **FIGURE 10–19**

Simplified block diagram of a 1M × 1 DRAM.

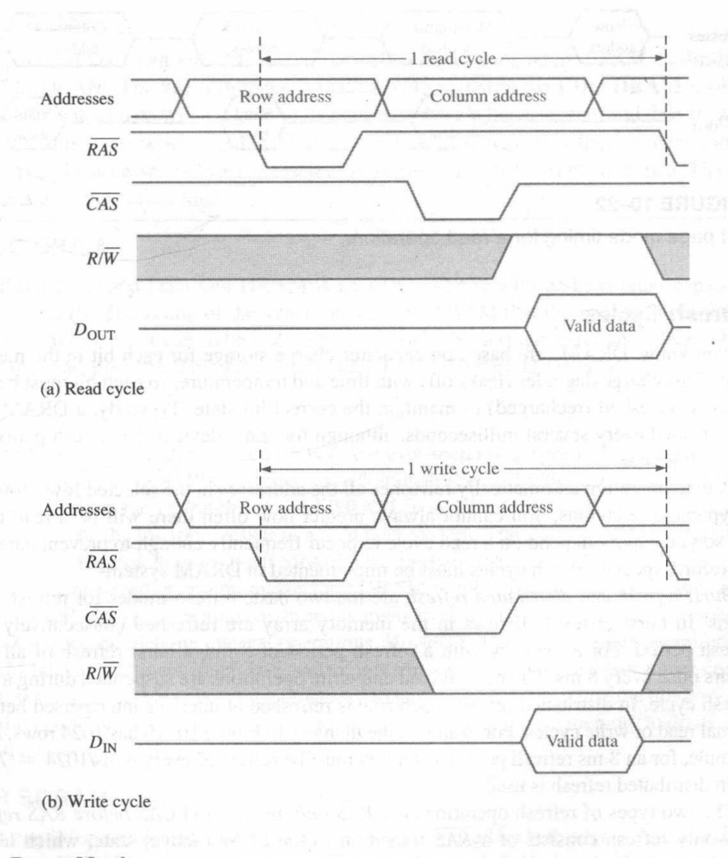
▶ **FIGURE 10–20**
Basic timing for address multiplexing.



Read and Write Cycles

At the beginning of each read or write memory cycle, $\overline{RAS}$ and $\overline{CAS}$ go active (LOW) to multiplex the row and column addresses into the registers, and decoders. For a read cycle, the $R/\overline{W}$ input is HIGH. For a write cycle, the $R/\overline{W}$ input is LOW. This is illustrated in Figure 10–21.

▶ **FIGURE 10–21**
Timing diagrams for normal read and write cycles.



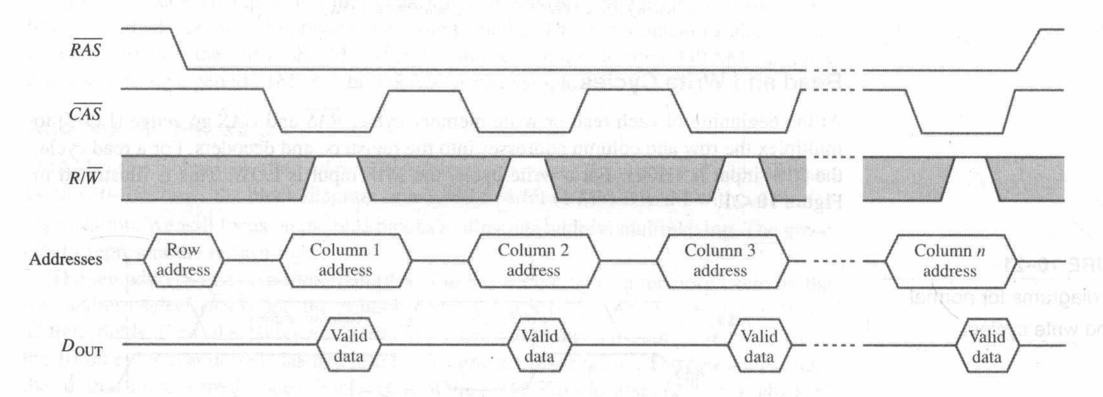
Fast Page Mode

In the normal read or write cycle described previously, the row address for a particular memory location is first loaded by an active-LOW $\overline{RAS}$ and then the column address for that location is loaded by an active-LOW $\overline{CAS}$. The next location is selected by another $\overline{RAS}$ followed by a $\overline{CAS}$, and so on.

A "page" is a section of memory available at a single row address and consists of all the columns in a row. Fast page mode allows fast successive read or write operations at each column address in a selected row. A row address is first loaded by $\overline{RAS}$ going LOW and remaining LOW while $\overline{CAS}$ is toggled between HIGH and LOW. A single row address is selected and remains selected while $\overline{RAS}$ is active. Each successive $\overline{CAS}$ selects another column in the selected row. So, after a fast page mode cycle, all of the addresses in the

selected row have been read from or written into, depending on $R/\overline{W}$. For example, a fast page mode cycle for the DRAM in Figure 10–19 requires $\overline{CAS}$ to go active 1024 times for each row selected by $\overline{RAS}$.

Fast page mode operation for read is illustrated by the timing diagram in Figure 10–22. When $\overline{CAS}$ goes to its nonasserted state (HIGH), it disables the data outputs. Therefore, the transition of $\overline{CAS}$ to HIGH must occur only after valid data are latched by the external system.



▲ FIGURE 10–22

Fast page mode timing for a read operation.

Refresh Cycles

As you know, DRAMs are based on capacitor charge storage for each bit in the memory array. This charge degrades (leaks off) with time and temperature, so each bit must be periodically refreshed (recharged) to maintain the correct bit state. Typically, a DRAM must be refreshed every several milliseconds, although for some devices the refresh period can be much longer.

A read operation automatically refreshes all the addresses in the selected row. However, in typical applications, you cannot always predict how often there will be a read cycle, and so you cannot depend on a read cycle to occur frequently enough to prevent data loss. Therefore, special refresh cycles must be implemented in DRAM systems.

Burst refresh and *distributed refresh* are the two basic refresh modes for refresh operations. In burst refresh, all rows in the memory array are refreshed consecutively each refresh period. For a memory with a refresh period of 8 ms, a burst refresh of all rows occurs once every 8 ms. The normal read and write operations are suspended during a burst refresh cycle. In distributed refresh, each row is refreshed at intervals interspersed between normal read or write cycles. For example, the memory in Figure 10–19 has 1024 rows. As an example, for an 8 ms refresh period, each row must be refreshed every 8 ms/1024 = 7.8 μs when distributed refresh is used.

The two types of refresh operations are *$\overline{RAS}$ only refresh* and *$\overline{CAS}$ before $\overline{RAS}$ refresh.* $\overline{RAS}$-only refresh consists of a $\overline{RAS}$ transition to the LOW (active) state, which latches the address of the row to be refreshed while $\overline{CAS}$ remains HIGH (inactive) throughout the cycle. An external counter is used to provide the row addresses for this type of operation.

The $\overline{CAS}$ before $\overline{RAS}$ refresh is initiated by $\overline{CAS}$ going LOW before $\overline{RAS}$ goes LOW. This sequence activates an internal refresh counter that generates the row address to be refreshed. This address is switched by the data selector into the row decoder.

Types of DRAMs

Now that you have learned the basic concept of a DRAM, let's briefly look at the major types. These are the *Fast Page Mode (FPM) DRAM,* the *Extended Data Out (EDO) DRAM,* the *Burst Extended Data Out (BEDO) DRAM,* and the *Synchronous (S) DRAM.*

FPM DRAM

Fast page mode operation was described earlier. Recall that a page in memory is all of the column addresses contained within one row address.

The idea of the **FPM DRAM** is based on the probability that the next several memory addresses to be accessed are in the same row (on the same page). Fortunately, this happens a large percentage of the time. FPM saves time over pure random accessing because in FPM the row address is specified only once for access to several successive column addresses whereas for pure random accessing, a row address is specified for each column address.

Recall that in a fast page mode read operation, the $\overline{CAS}$ signal has to wait until the valid data from a given address are accepted (latched) by the external system (CPU) before it can go to its nonasserted state. When $\overline{CAS}$ goes to its nonasserted state, the data outputs are disabled. This means that the next column address cannot occur until after the data from the current column address are transferred to the CPU. This limits the rate at which the columns within a page can be addressed.

EDO DRAM

The Extended Data Out DRAM, sometimes called *hyper page mode DRAM,* is similar to the FPM DRAM. The key difference is that the $\overline{CAS}$ signal in the **EDO DRAM** does not disable the output data when it goes to its nonasserted state because the valid data from the current address can be held until $\overline{CAS}$ is asserted again. This means that the next column address can be accessed before the external system accepts the current valid data. The idea is to speed up the access time.

BEDO DRAM

The Burst Extended Data Out DRAM is an EDO DRAM with address burst capability. Recall from the discussion of the synchronous burst SRAM that the address burst feature allows up to four addresses to be internally generated from a single external address, which saves some access time. This same concept applies to the **BEDO DRAM.**

SDRAM

Faster DRAMs are needed to keep up with the ever-increasing speed of microprocessors. The Synchronous DRAM is one way to accomplish this. Like the synchronous SRAM discussed earlier, the operation of the **SDRAM** is synchronized with the system clock, which also runs the microprocessor in a computer system. The same basic ideas described in relation to the synchronous burst SRAM, also apply to the SDRAM.

This synchronized operation makes the SDRAM totally different from the other asynchronous DRAM types. With asynchronous memories, the microprocessor must wait for the DRAM to complete its internal operations. However, with synchronous operation, the DRAM latches addresses, data, and control information from the processor under control of the system clock. This allows the processor to handle other tasks while the memory read or write operations are in progress, rather than having to wait for the memory to do its thing as is the case in asynchronous systems.

DDR SDRAM

DDR stands for double data rate. A DDR SDRAM is clocked on both edges of a clock pulse, whereas a SDRAM is clocked on only one edge. Because of the double clocking, a DDR SDRAM is theoretically twice as fast as an SDRAM. Sometimes the SDRAM is referred to as an SDR SDRAM (single data rate SDRAM) for contrast with the DDR SDRAM.

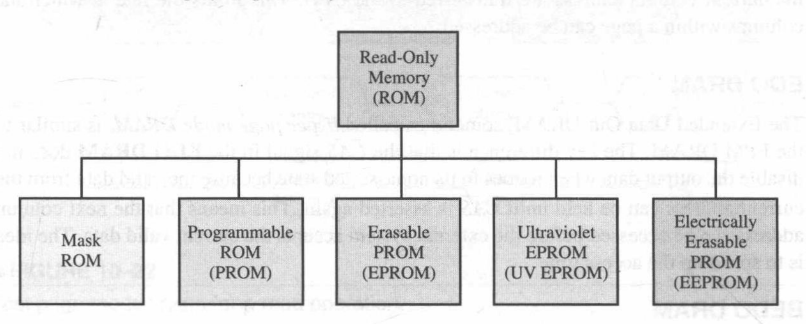
SECTION 10-2 CHECKUP

1. List two types of SRAM.
2. What is a cache?
3. Explain how SRAMs and DRAMs differ.
4. Describe the refresh operation in a DRAM.
5. List four types of DRAM.

10-3 The Read-Only Memory (ROM)

The ROM Family

Figure 10–23 shows how semiconductor ROMs are categorized. The mask ROM is the type in which the data are permanently stored in the memory during the manufacturing process. The **PROM,** or programmable ROM, is the type in which the data are electrically stored by the user with the aid of specialized equipment. Both the mask ROM and the PROM can be of either MOS or bipolar technology. The **EPROM,** or erasable PROM, is strictly a MOS device. The **UV EPROM** is electrically programmable by the user, but the stored data must be erased by exposure to ultraviolet light over a period of several minutes. The electrically erasable PROM (**EEPROM** or E^2PROM) can be erased in a few milliseconds. The UV EPROM has been largely displaced by the EEPROM.

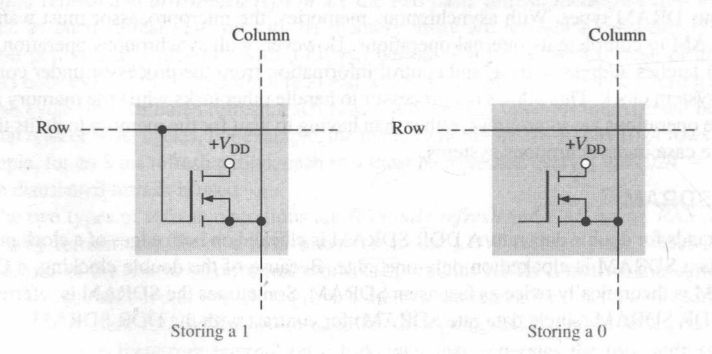


◄ **FIGURE 10–23**
The ROM family.

The Mask ROM

The mask ROM is usually referred to simply as a ROM. It is permanently programmed during the manufacturing process to provide widely used standard functions, such as popular conversions, or to provide user-specified functions. Once the memory is programmed, it cannot be changed. Most IC ROMs utilize the presence or absence of a transistor connection at a row/column junction to represent a 1 or a 0.

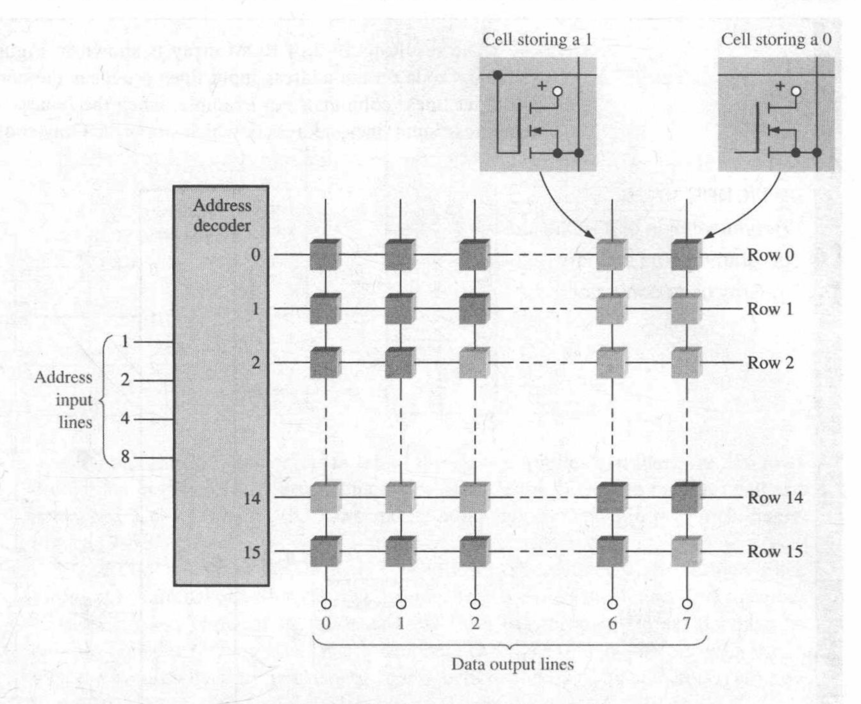
Figure 10–24 shows MOS ROM cells. The presence of a connection from a row line to the gate of a transistor represents a 1 at that location because when the row line is taken HIGH, all transistors with a gate connection to that row line turn on and connect the HIGH (1) to the associated column lines. At row/column junctions where there are no gate connections, the column lines remain LOW (0) when the row is addressed.



◄ **FIGURE 10–24**
ROM cells.

To illustrate the ROM concept, Figure 10–25 shows a small, simplified ROM array. The blue squares represent stored 1s, and the gray squares represent stored 0s. The basic read operation is as follows. When a binary address code is applied to the address input lines, the corresponding row line goes HIGH. This HIGH is connected to the column lines through the transistors at each junction (cell) where a 1 is stored. At each cell where a 0 is stored, the column line stays LOW because of the terminating resistor. The column lines form the data output. The eight data bits stored in the selected row appear on the output lines.

▶ **FIGURE 10–25**
A representation of a 16 × 8-bit ROM array.



As you can see, the example ROM in Figure 10–25 is organized into 16 addresses, each of which stores 8 data bits. Thus, it is a 16 × 8 (16-by-8) ROM, and its total capacity is 128 bits or 16 bytes. ROMs can be used as look-up tables (LUTs) for code conversions and logic function generation.

EXAMPLE 10-1

Show a basic ROM, similar to the one in Figure 10–25, programmed for a 4-bit binary-to-Gray conversion.

Solution

Review Chapter 2 for the Gray code. Table 10–1 is developed for use in programming the ROM.

▶ **TABLE 10–1**

Binary				Gray			
B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

The resulting 16 × 4 ROM array is shown in Figure 10–26. You can see that a binary code on the address input lines produces the corresponding Gray code on the output lines (columns). For example, when the binary number 0110 is applied to the address input lines, address 6, which stores the Gray code 0101, is selected.

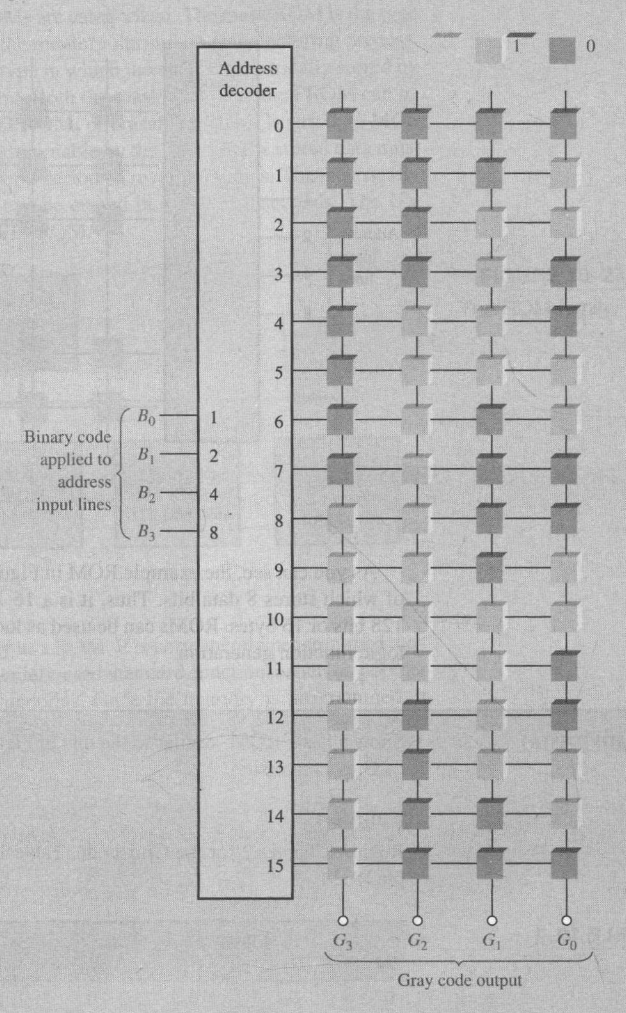


FIGURE 10–26
Representation of a ROM programmed as a binary-to-Gray code converter.

Related Problem*

Using Figure 10–26, determine the Gray code output when a binary code of 1011 is applied to the address input lines.

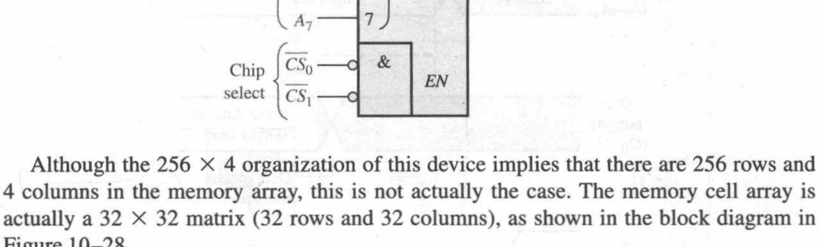
*Answers are at the end of the chapter.

Internal ROM Organization

Most IC ROMs have a more complex internal organization than that in the basic simplified example just presented. To illustrate how an IC ROM is structured, let's use a 1024-bit device with a 256 × 4 organization. The logic symbol is shown in Figure 10–27. When any one of 256 binary codes (eight bits) is applied to the address lines, four data bits appear on the outputs if the chip select inputs are LOW. (256 addresses require eight address lines.)

▶ **FIGURE 10–27**

A 256 × 4 ROM logic symbol. The $A\frac{0}{255}$ designator means that the 8-bit address code selects addresses 0 through 255.



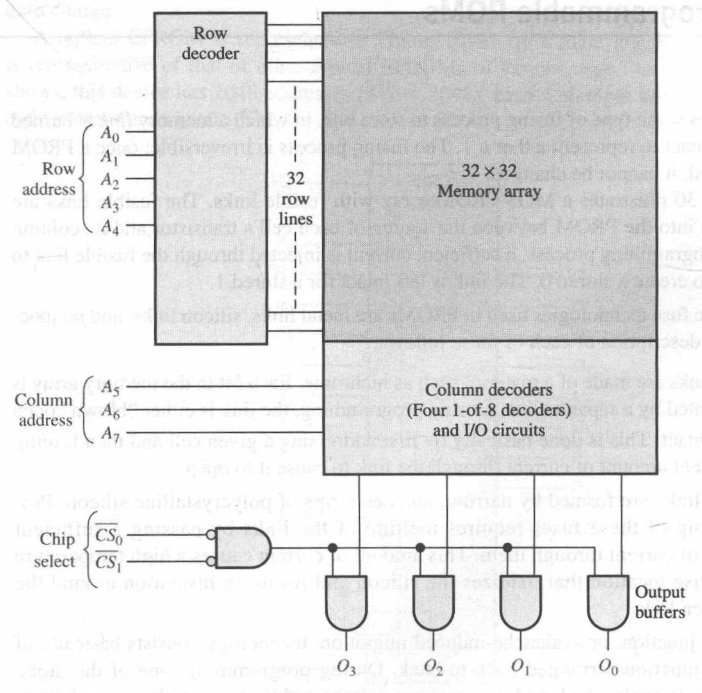
Although the 256 × 4 organization of this device implies that there are 256 rows and 4 columns in the memory array, this is not actually the case. The memory cell array is actually a 32 × 32 matrix (32 rows and 32 columns), as shown in the block diagram in Figure 10–28.

The ROM in Figure 10–28 works as follows. Five of the eight address lines (A_0 through A_4) are decoded by the row decoder (often called the Y decoder) to select one of the 32 rows. Three of the eight address lines (A_5 through A_7) are decoded by the column decoder (often called the X decoder) to select four of the 32 columns. Actually, the column decoder consists of four 1-of-8 decoders (data selectors), as shown in Figure 10–28.

The result of this structure is that when an 8-bit address code (A_0 through A_7) is applied, a 4-bit data word appears on the data outputs when the chip select lines ($\overline{CS_0}$ and $\overline{CS_1}$) are LOW to enable the output buffers. This type of internal organization (architecture) is typical of IC ROMs of various capacities.

InfoNote

ROM is used in a computer to store the BIOS (Basic Input/Output System). These are programs that are used to perform fundamental supervisory and support functions for the computer. For example, BIOS programs stored in the ROM control certain video monitor functions, provide for disk formatting, scan the keyboard for inputs, and control certain printer functions.



▲ **FIGURE 10–28**

A 1024-bit ROM with a 256 × 4 organization based on a 32 × 32 array.

ROM Access Time

A typical timing diagram that illustrates ROM access time is shown in Figure 10–29. The **access time,** t_a, of a ROM is the time from the application of a valid address code on the input lines until the appearance of valid output data. Access time can also be measured from the activation of the chip select ($\overline{CS}$) input to the occurrence of valid output data when a valid address is already on the input lines.

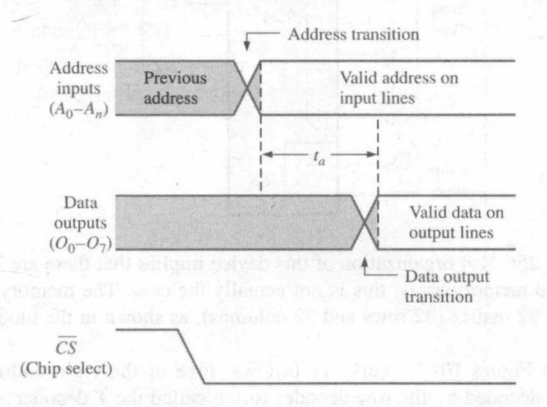


FIGURE 10–29 ROM access time (t_a) from address change to data output with chip select already active.

SECTION 10-3 CHECKUP

1. What is the bit storage capacity of a ROM with a 512 × 8 organization?
2. List the types of read-only memories.
3. How many address bits are required for a 2048-bit memory organized as a 256 × 8 memory?

10-4 Programmable ROMs

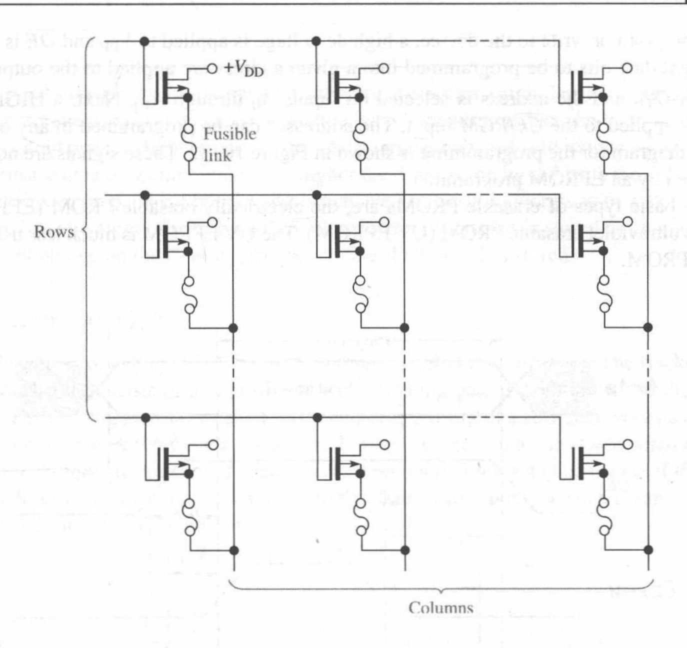
PROMs

A **PROM** uses some type of fusing process to store bits, in which a memory *link* is burned open or left intact to represent a 0 or a 1. The fusing process is irreversible; once a PROM is programmed, it cannot be changed.

Figure 10–30 illustrates a MOS PROM array with fusible links. The fusible links are manufactured into the PROM between the source of each cell's transistor and its column line. In the programming process, a sufficient current is injected through the fusible link to burn it open to create a stored 0. The link is left intact for a stored 1.

Three basic fuse technologies used in PROMs are metal links, silicon links, and *pn* junctions. A brief description of each of these follows.

1. Metal links are made of a material such as nichrome. Each bit in the memory array is represented by a separate link. During programming, the link is either "blown" open or left intact. This is done basically by first addressing a given cell and then forcing a sufficient amount of current through the link to cause it to open.
2. Silicon links are formed by narrow, notched strips of polycrystalline silicon. Programming of these fuses requires melting of the links by passing a sufficient amount of current through them. This amount of current causes a high temperature at the fuse location that oxidizes the silicon and forms an insulation around the now-open link.
3. Shorted junction, or avalanche-induced migration, technology consists basically of two *pn* junctions arranged back-to-back. During programming, one of the diode junctions is avalanched, and the resulting voltage and heat cause aluminum ions to migrate and short the junction. The remaining junction is then used as a forward-biased diode to represent a data bit.



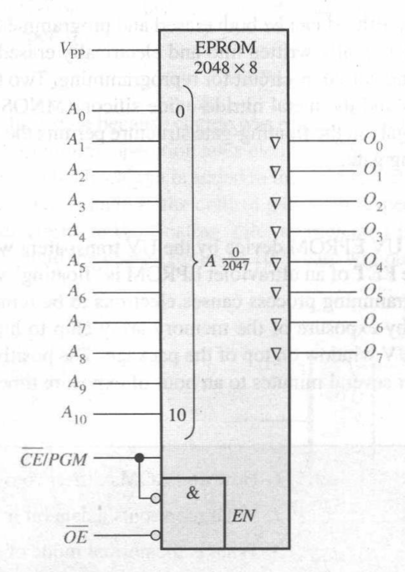
▶ **FIGURE 10–30**
MOS PROM array with fusible links. (All drains are commonly connected to V_{DD}.)

EPROMs

An **EPROM** is an erasable PROM. Unlike an ordinary PROM, an EPROM can be reprogrammed if an existing program in the memory array is erased first.

An EPROM uses an NMOSFET array with an isolated-gate structure. The isolated transistor gate has no electrical connections and can store an electrical charge for indefinite periods of time. The data bits in this type of array are represented by the presence or absence of a stored gate charge. Erasure of a data bit is a process that removes the gate charge.

A typical EPROM is represented in Figure 10–31 by a logic diagram. Its operation is representative of that of other typical EPROMs of various sizes. As the logic symbol shows, this device has 2048 addresses ($2^{11} = 2048$), each with eight bits. Notice that the eight outputs are tri-state (∇).



▶ **FIGURE 10–31**
The logic symbol for a 2048 × 8 EPROM.

To read from the memory, the output enable input ($\overline{OE}$) must be LOW and the power-down/program ($\overline{CE}/PGM$) input LOW.

To program or write to the device, a high dc voltage is applied to V_{PP} and $\overline{OE}$ is HIGH. The eight data bits to be programmed into a given address are applied to the outputs (O_0 through O_7), and the address is selected on inputs A_0 through A_{10}. Next, a HIGH level pulse is applied to the $\overline{CE}/PGM$ input. The addresses can be programmed in any order. A timing diagram for the programming is shown in Figure 10–32. These signals are normally produced by an EPROM programmer.

Two basic types of erasable PROMs are, the electrically erasable PROM (EEPROM) and the ultraviolet erasable PROM (UV EPROM). The UV EPROM is much less used than the EEPROM.

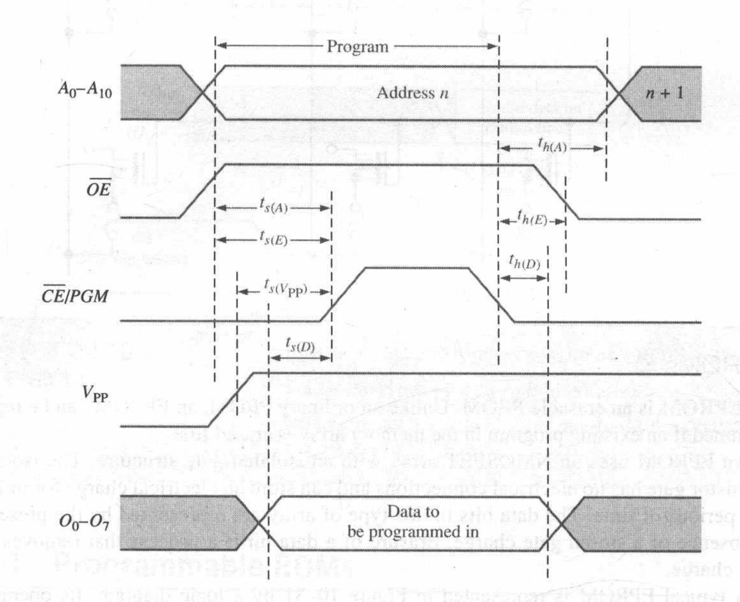


FIGURE 10–32 Timing diagram for a 2048 × 8 EPROM programming cycle, with critical setup times (t_s) and hold times (t_h) indicated.

EEPROMs

An electrically erasable PROM can be both erased and programmed with electrical pulses. Since it can be both electrically written into and electrically erased, the EEPROM can be rapidly programmed and erased in-circuit for reprogramming. Two types of EEPROMs are the floating-gate MOS and the metal nitride-oxide silicon (MNOS). The application of a voltage on the control gate in the floating-gate structure permits the storage and removal of charge from the floating gate.

UV EPROMs

You can recognize the UV EPROM device by the UV transparent window on the package. The isolated gate in the **FET** of an ultraviolet EPROM is "floating" within an oxide insulating material. The programming process causes electrons to be removed from the floating gate. Erasure is done by exposure of the memory array chip to high-intensity ultraviolet radiation through the UV window on top of the package. The positive charge stored on the gate is neutralized after several minutes to an hour of exposure time.

SECTION 10-4 CHECKUP

1. How do PROMs differ from ROMs?
2. What represents a data bit in an EPROM?
3. What is the normal mode of operation for a PROM?

10-5 The Flash Memory

Flash memories are high-density read/write memories (high-density translates into large bit storage capacity) that are nonvolatile, which means that data can be stored indefinitely without power. High-density means that a large number of cells can be packed into a given surface area on a chip; that is, the higher the density, the more bits that can be stored on a given size chip. This high density is achieved in flash memories with a storage cell that consists of a single floating-gate MOS transistor. A data bit is stored as charge or the absence of charge on the floating gate depending if a 0 or a 1 is stored.

Flash Memory Cell

A single-transistor cell in a flash memory is represented in Figure 10–33. The stacked gate MOS transistor consists of a control gate and a floating gate in addition to the drain and source. The floating gate stores electrons (charge) as a result of a sufficient voltage applied to the control gate. A *0 is stored when there is more charge* and a *1 is stored when there is less or no charge.* The amount of charge present on the floating gate determines if the transistor will turn on and conduct current from the drain to the source when a control voltage is applied during a read operation.

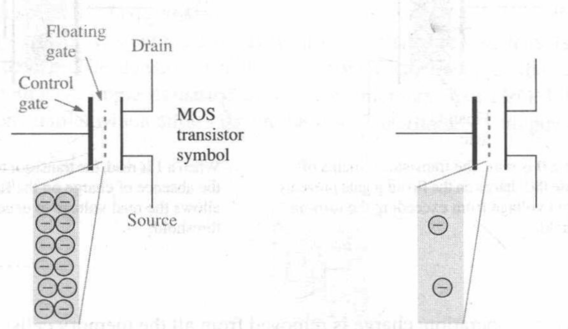


FIGURE 10–33
The storage cell in a flash memory.

Basic Flash Memory Operation

There are three major operations in a flash memory: the *programming* operation, the *read* operation, and the *erase* operation.

Programming

Initially, all cells are at the 1 state because charge was removed from each cell in a previous erase operation. The programming operation adds electrons (charge) to the floating gate of those cells that are to store a 0. No charge is added to those cells that are to store a 1. Application of a sufficient positive voltage to the control gate with respect to the source during programming attracts electrons to the floating gate, as indicated in Figure 10–34. Once programmed, a cell can retain the charge for up to 100 years without any external power.

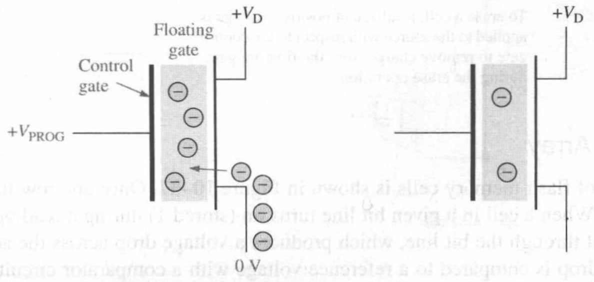
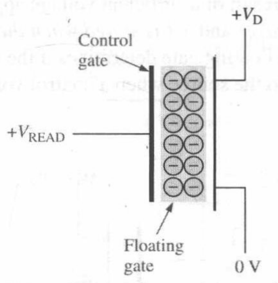


FIGURE 10–34
Simplified illustration of storing a 0 or a 1 in a flash cell during the programming operation.

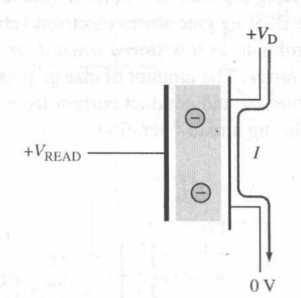
Read

During a read operation, a positive voltage is applied to the control gate. The amount of charge present on the floating gate of a cell determines whether or not the voltage applied to the control gate will turn on the transistor. If a 1 is stored, the control gate voltage is sufficient to turn the transistor on. If a 0 is stored, the transistor will not turn on because the control gate voltage is not sufficient to overcome the negative charge stored in the floating gate. Think of the charge on the floating gate as a voltage source that opposes the voltage applied to the control gate during a read operation. So the floating gate charge associated with a stored 0 prevents the control gate voltage from reaching the turn-on threshold, whereas the small or zero charge associated with a stored 1 allows the control gate voltage to exceed the turn-on threshold.

When the transistor turns on, there is current from the drain to the source of the cell transistor. The presence of this current is sensed to indicate a 1, and the absence of this current is sensed to indicate a 0. This basic idea is illustrated in Figure 10–35.



When a 0 is read, the transistor remains off because the charge on the floating gate prevents the read voltage from exceeding the turn-on threshold.



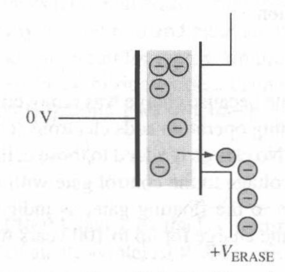
When a 1 is read, the transistor turns on because the absence of charge on the floating gate allows the read voltage to exceed the turn-on threshold.

FIGURE 10–35

The read operation of a flash cell in an array.

Erase

During an erase operation, charge is removed from all the memory cells. A sufficient positive voltage is applied to the transistor source with respect to the control gate. This is opposite in polarity to that used in programming. This voltage attracts electrons from the floating gate and depletes it of charge, as illustrated in Figure 10–36. A flash memory is always erased prior to being reprogrammed.



To erase a cell, a sufficient positive voltage is applied to the source with respect to the control gate to remove charge from the floating gate during the erase operation.

FIGURE 10–36

Simplified illustration of removing charge from a cell during erase.

Flash Memory Array

A simplified array of flash memory cells is shown in Figure 10–37. Only one row line is accessed at a time. When a cell in a given bit line turns on (stored 1) during a read operation, there is current through the bit line, which produces a voltage drop across the active load. This voltage drop is compared to a reference voltage with a comparator circuit and an output level indicating a 1 is produced. If a 0 is stored, then there is no current or little current in the bit line and an opposite level is produced on the comparator output.

The memory stick is a storage medium that uses flash memory technology in a physical configuration smaller than a stick of chewing gum. Memory sticks are typically available up to 64 GB capacities and as a kit with a PC card adaptor. Because of its compact design, it is ideal for use in small digital electronics products, such as laptop computers and digital cameras.

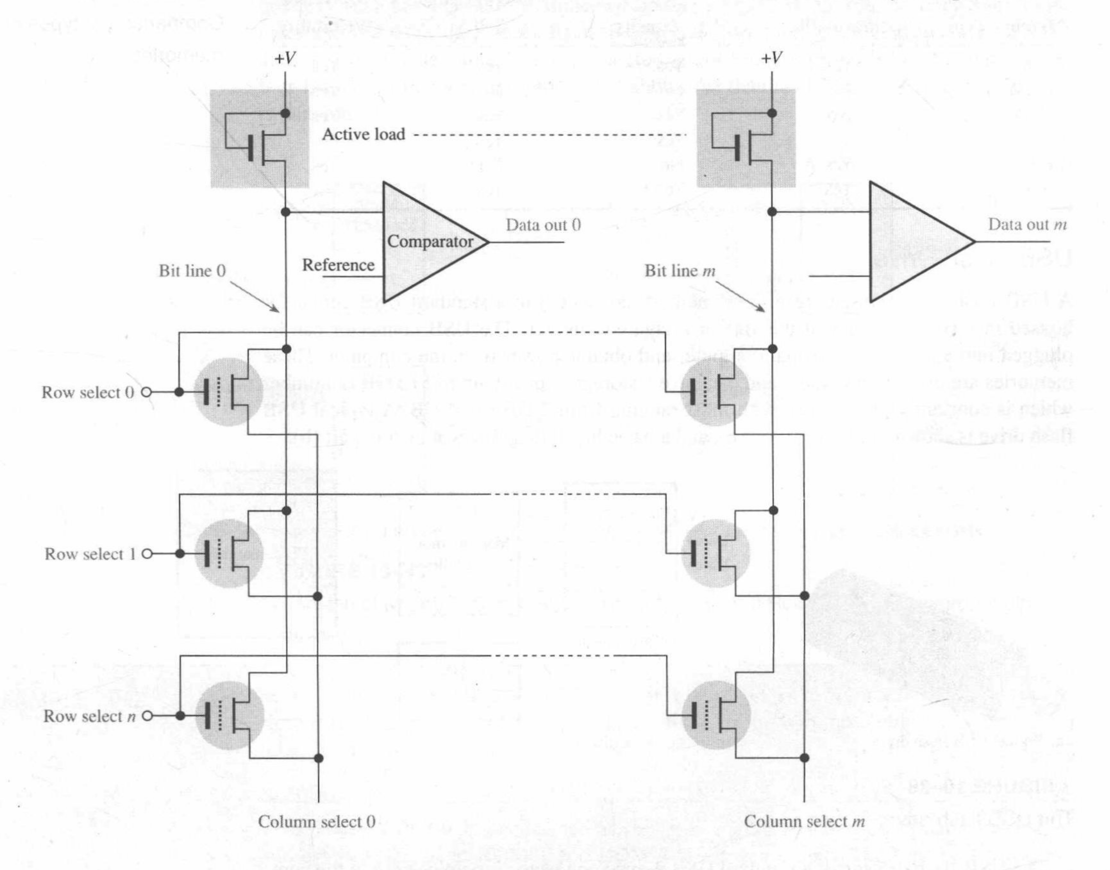
Comparison of Flash Memories with Other Memories

Let's compare flash memories with other types of memories with which you are already familiar.

Flash vs. ROM, EPROM, and EEPROM

Read-only memories are high-density, nonvolatile devices. However, once programmed the contents of a ROM can never be altered. Also, the initial programming is a time-consuming and costly process. The EEPROM has a more complex cell structure than either the ROM or UV EPROM and so the density is not as high, although it can be reprogrammed without being removed from the system. Because of its lower density, the cost/bit is higher than ROMs or EPROMs. Although the UV EPROM is a high-density, nonvolatile memory, it can be erased only by removing it from the system and using ultraviolet light. It can be reprogrammed only with specialized equipment.

A flash memory can be reprogrammed easily in the system because it is essentially a READ/WRITE device. The density of a flash memory compares with the ROM and EPROM because both have single-transistor cells. A flash memory (like a ROM, EPROM, or EEPROM) is nonvolatile, which allows data to be stored indefinitely with power off.



▲FIGURE 10–37
Basic flash memory array.

Flash vs. SRAM

As you have learned, static random-access memories are volatile READ/WRITE devices. A SRAM requires constant power to retain the stored data. In many applications, a battery backup is used to prevent data loss if the main power source is turned off. However, since battery failure is always a possibility, indefinite retention of the stored data in a SRAM cannot be guaranteed. Because the memory cell in a SRAM is basically a flip-flop consisting of several transistors, the density is relatively low.

A flash memory is also a READ/WRITE memory, but unlike the SRAM it is nonvolatile. Also, a flash memory has a much higher density than a SRAM.

Flash vs. DRAM

Dynamic random-access memories are volatile high-density READ/WRITE devices. DRAMs require not only constant power to retain data but also that the stored data must be refreshed frequently. In many applications, backup storage such as hard disk must be used with a DRAM.

Flash memories exhibit higher densities than DRAMs because a flash memory cell consists of one transistor and does not need refreshing, whereas a DRAM cell is one transistor plus a capacitor that has to be refreshed. Typically, a flash memory consumes much less power than an equivalent DRAM and can be used as a hard disk replacement in many applications.

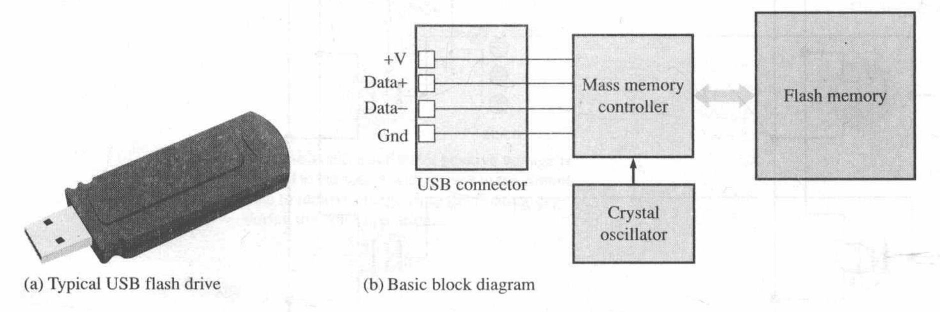
Table 10–2 provides a comparison of the memory technologies.

TABLE 10–2
Comparison of types of memories.

Memory Type	Nonvolatile	High-Density	One-Transistor Cell	In-System Writability
Flash	Yes	Yes	Yes	Yes
SRAM	No	No	No	Yes
DRAM	No	Yes	Yes	Yes
ROM	Yes	Yes	Yes	No
EEPROM	Yes	No	No	Yes
UV EPROM	Yes	Yes	Yes	No

USB Flash Drive

A USB flash drive consists of a flash memory connected to a standard USB connector housed in a small case about the size of a cigarette lighter. The USB connector can be plugged into a port on a personal computer and obtains power from the computer. These memories are usually rewritable and can have a storage capacity up to 512 GB (a number which is constantly increasing), with most ranging from 2 GB to 64 GB. A typical USB flash drive is shown in Figure 10–38(a), and a basic block diagram is shown in part (b).



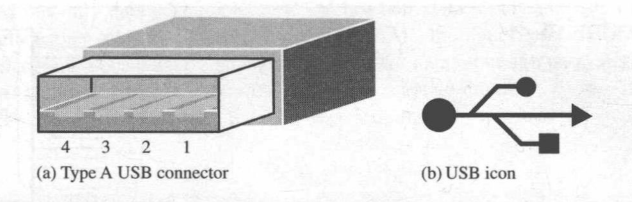
(a) Typical USB flash drive (b) Basic block diagram

FIGURE 10–38
The USB flash drive.

The USB flash drive uses a standard USB A-type connector for connection to the computer, as shown in Figure 10–39(a). Peripherals such as printers use the USB B-type connector, which has a different shape and physical pin configuration. The USB icon is shown in part (b).

FIGURE 10–39

Connector and symbol.



(a) Type A USB connector (b) USB icon

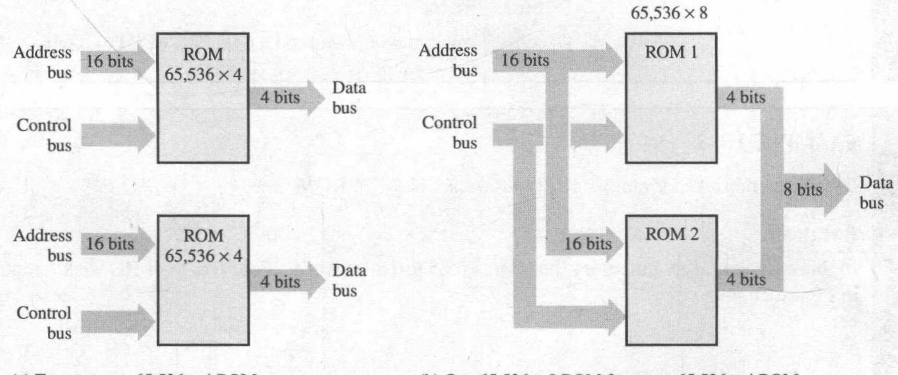
SECTION 10-5 CHECKUP

1. What types of memories are nonvolatile?
2. What is a major advantage of a flash memory over a SRAM or DRAM?
3. List the three modes of operation of a flash memory.

10-6 Memory Expansion

Word-Length Expansion

To increase the **word length** of a memory, the number of bits in the data bus must be increased. For example, an 8-bit word length can be achieved by using two memories, each with 4-bit words as illustrated in Figure 10–40(a). As you can see in part (b), the 16-bit address bus is commonly connected to both memories so that the combination memory still has the same number of addresses ($2^{16} = 65{,}536$) as each individual memory. The 4-bit data buses from the two memories are combined to form an 8-bit data bus. Now when an address is selected, eight bits are produced on the data bus—four from each memory. Example 10–2 shows the details of 65,536 × 4 to 65,536 × 8 expansion.



(a) Two separate 65,536 × 4 ROMs (b) One 65,536 × 8 ROM from two 65,536 × 4 ROMs

▲ FIGURE 10–40

Expansion of two 65,536 × 4 ROMs into a 65,536 × 8 ROM to illustrate word-length expansion.

EXAMPLE 10-2

Expand the 65,536 × 4 ROM (64k × 4) in Figure 10–41 to form a 64k × 8 ROM. Note that "64k" is the accepted shorthand for 65,536. Why not "65k"? Maybe it's because 64 is also a power-of-two.

Solution

Two 64k × 4 ROMs are connected as shown in Figure 10–42. Notice that a specific address is accessed in ROM 1 and ROM 2 at the same time. The four bits from a selected address in ROM 1 and the four bits from the corresponding address in ROM 2 go out in parallel to form an 8-bit word on the data bus. Also notice that a LOW on the enable line, $\overline{E}$, which forms a simple control bus, enables *both* memories.

FIGURE 10–41

A 64k × 4 ROM.

FIGURE 10–42

Related Problem

Describe how you would expand a 64k × 1 ROM to a 64k × 8 ROM.

EXAMPLE 10-3

Use the memories in Example 10–2 to form a 64k × 16 ROM.

Solution

In this case you need a memory that stores 65,536 16-bit words. Four 64k × 4 ROMs are required to do the job, as shown in Figure 10–43.

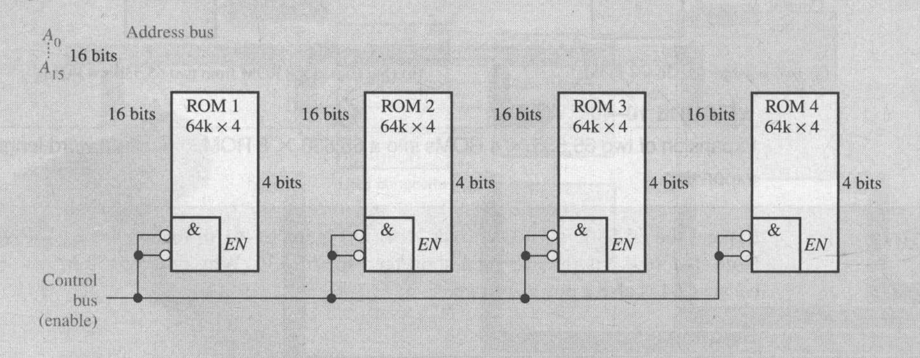


FIGURE 10–43

Related Problem

How many 64k × 1 ROMs would be required to implement the memory shown in Figure 10–43?

A ROM has only data outputs, but a RAM has both data inputs and data outputs. For word-length expansion in a RAM (SRAM or DRAM), the data inputs *and* data outputs form the data bus. Because the same lines are used for data input and data output, tri-state buffers are required. Most RAMs provide internal tri-state circuitry. Figure 10–44 illustrates RAM expansion to increase the data word length.

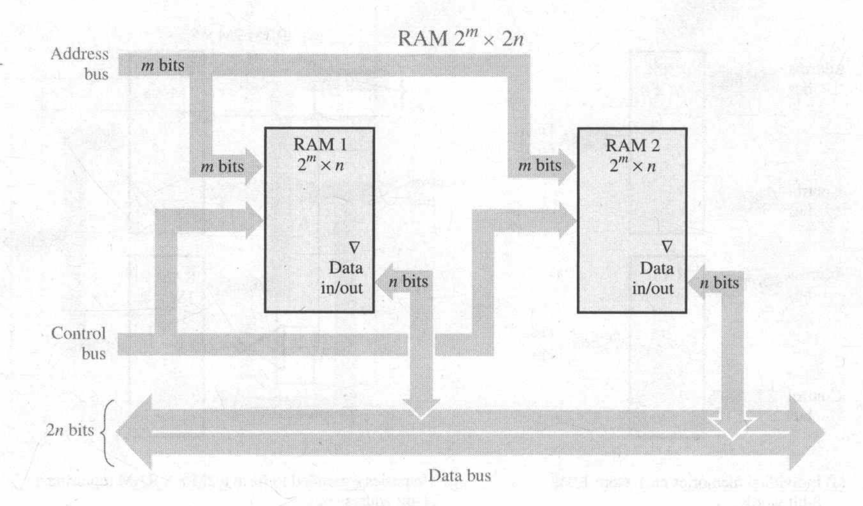


FIGURE 10–44 Illustration of word-length expansion with two $2^m \times n$ RAMs forming a $2^m \times 2n$ RAM.

EXAMPLE 10-4

Use 1M × 4 SRAMs to create a 1M × 8 SRAM.

Solution

Two 1M × 4 SRAMs are connected as shown in the simplified block diagram of Figure 10–45.

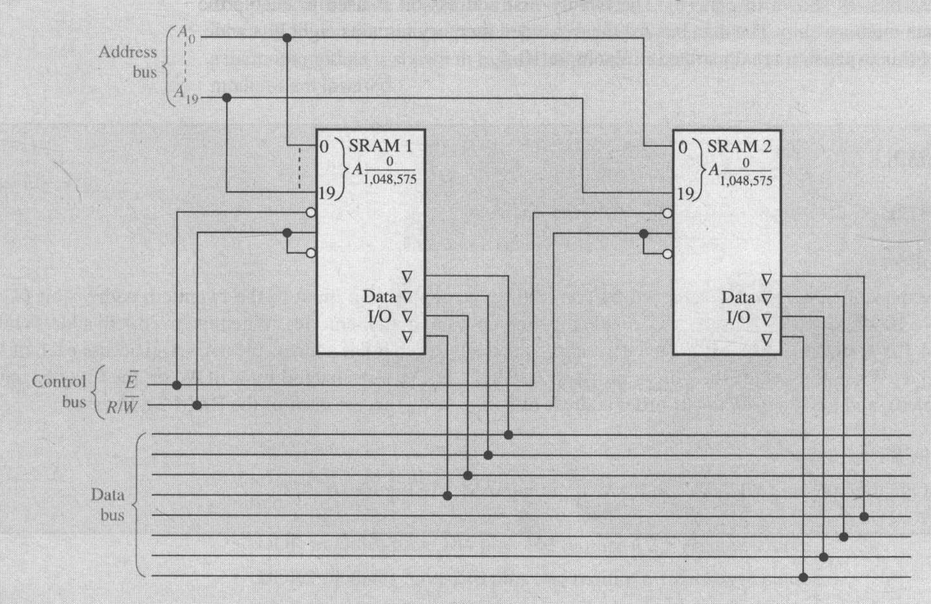


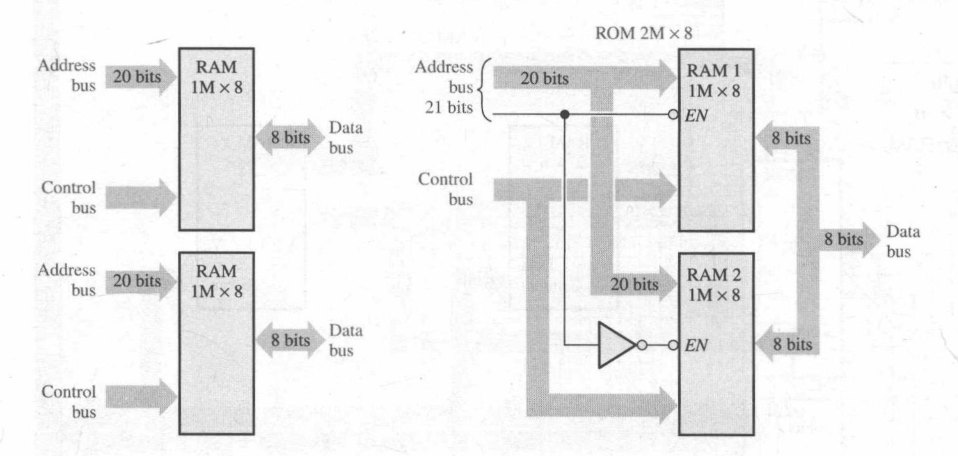
FIGURE 10–45

Related Problem

Use 1M × 8 SRAMs to create a 1M × 16 SRAM.

Word-Capacity Expansion

When memories are expanded to increase the **word capacity**, the *number of addresses is increased.* To achieve this increase, the number of address bits must be increased, as illustrated in Figure 10–46, (where two 1M × 8 RAMs are expanded to form a 2M × 8 memory).



(a) Individual memories each store 1,048,576 8-bit words

(b) Memories expanded to form a 2M × 8 RAM requiring a 21-bit address bus

▲ FIGURE 10–46

Illustration of word-capacity expansion.

Each individual memory has 20 address bits to select its 1,048,576 addresses, as shown in part (a). The expanded memory has 2,097,152 addresses and therefore requires 21 address bits, as shown in part (b). The twenty-first address bit is used to enable the appropriate memory chip. The data bus for the expanded memory remains eight bits wide. Details of this expansion are illustrated in Example 10–5.

EXAMPLE 10-5

Use 512k × 4 RAMs to implement a 1M × 4 memory.

Solution

The expanded addressing is achieved by connecting the enable ($\overline{E}_0$) input to the twentieth address bit (A_{19}), as shown in Figure 10–47. Input $\overline{E}_1$ is used as an enable input common to both memories. When the twentieth address bit (A_{19}) is LOW, RAM 1 is selected (RAM 2 is disabled), and the nineteen lower-order address bits (A_0–A_{18}) access each of the addresses in RAM 1. When the twentieth address bit (A_{19}) is HIGH, RAM 2 is enabled by a LOW on the inverter output (RAM 1 is disabled), and the nineteen lower-order address bits ($A_0 - A_{18}$) access each of the RAM 2 addresses.

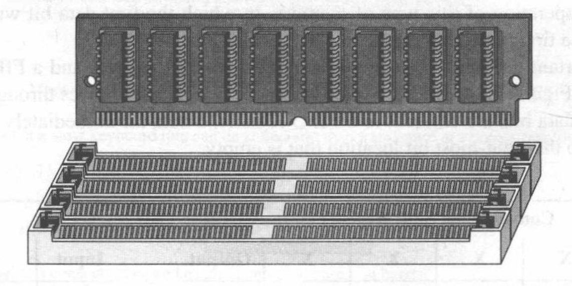
Related Problem

What are the ranges of addresses in RAM 1 and in RAM 2 in Figure 10–47?

▲FIGURE 10–47

Memory Modules

SDRAMs are available in modules consisting of multiple memory ICs arranged on a printed circuit board (PCB). The most common type of SDRAM memory module is called a **DIMM** (dual in-line memory module). Another version of the DIMM is the SODIMM (small-outline DIMM). A type of memory module, generally found in older equipment and essentially obsolete, is the **SIMM** (single in-line memory module). The SIMM has connection pins on one side of a PCB where the DIMM uses both sides of the board. DIMMs plug into a socket on the system mother board for memory expansion. A generic representation of a memory module is shown in Figure 10–48 with the system board connectors into which the modules are inserted.



►FIGURE 10–48
A memory module with connectors.

DIMMs generally contain DDR SDRAM memory chips. DDR means double data rate, so a DDR SDRAM transfers two blocks of data for each clock cycle rather than one like a standard SDRAM. Three basic types of modules are DDR, DDR2, and DDR3.

- DDR modules have 184 pins and require a 2.5 voltage source.
- DDR2 modules have 240 pins and require a 1.8 voltage source.
- DDR3 modules have 240 pins and require a 1.5 voltage source.

The DDR, DDR2, and DDR3 have transfer data rates of 1600 MB/s, 3200 MB/s, and 6400 MB/s respectively.

Memory components are extremely sensitive to static electricity. Use the following precautions when handling memory chips or modules such as DIMMs:

- Before handling, discharge your body's static charge by touching a grounded surface or wear a grounding wrist strap containing a high-value resistor if available. A convenient, reliable ground is the ac outlet ground.
- Do not remove components from their antistatic bags until you are ready to install them.
- Do not lay parts on the antistatic bags because only the inside is antistatic.
- When handling DIMMs, hold by the edges or the metal mounting bracket. Do not touch components on the boards or the edge connector pins.
- Never slide any part over any type of surface.
- Avoid plastic, vinyl, styrofoam, and nylon in the work area.

When installing DIMMs, follow these steps:

1. Line up the notches on the DIMM board with the notches in the memory socket.
2. Push firmly on the module until it is securely seated in the socket.
3. Generally, the latches on both sides of the socket will snap into place when the module is completely inserted. These latches also release the module, so it can be removed from the socket.

SECTION 10-6 CHECKUP

1. How many 16k × 1 RAMs are required to achieve a memory with a word capacity of 16k and a word length of eight bits?
2. To expand the 16k × 8 memory in question 1 to a 32k × 8 organization, how many more 16k × 1 RAMs are required?
3. What does DIMM stand for?

10-7 Special Types of Memories

First In–First Out (FIFO) Memories

This type of memory is formed by an arrangement of shift registers. The term **FIFO** refers to the basic operation of this type of memory, in which the first data bit written into the memory is the first to be read out.

One important difference between a conventional shift register and a FIFO register is illustrated in Figure 10–49. In a conventional register, a data bit moves through the register only as new data bits are entered; in a FIFO register, a data bit immediately goes through the register to the right-most bit location that is empty.

Conventional Shift Register

Input	X	X	X	X	Output
0	0	X	X	X	→
1	1	0	X	X	→
1	1	1	0	X	→
0	0	1	1	1	→

X = unknown data bits.
In a conventional shift register, data stay to the left until "forced" through by additional data.

FIFO Shift Register

Input	—	—	—	—	Output
0	—	—	—	0	→
1	—	—	1	0	→
1	—	1	1	0	→
0	0	1	1	0	→

— = empty positions.
In a FIFO shift register, data "fall" through (go right).

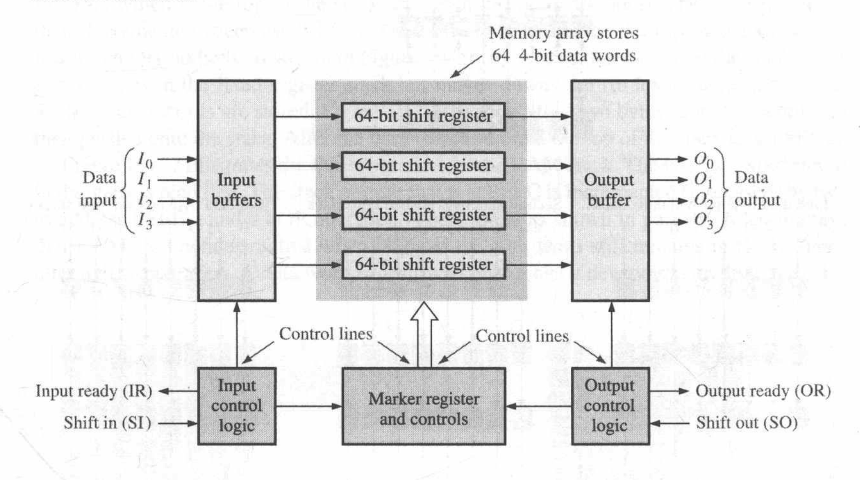
▲ **FIGURE 10–49**

Comparison of conventional and FIFO register operation.

Figure 10–50 is a block diagram of a FIFO serial memory. This particular memory has four serial 64-bit data registers and a 64-bit control register (marker register). When data are entered by a shift-in pulse, they move automatically under control of the marker register to the empty location closest to the output. Data cannot advance into occupied positions. However, when a data bit is shifted out by a shift-out pulse, the data bits remaining in the registers automatically move to the next position toward the output. In an asynchronous FIFO, data are shifted out independent of data entry, with the use of two separate clocks.

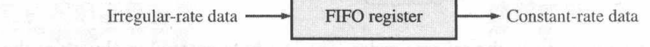
FIFO Applications

One important application area for the FIFO register is the case in which two systems of differing data rates must communicate. Data can be entered into a FIFO register at one rate and taken out at another rate. Figure 10–51 illustrates how a FIFO register might be used in these situations.

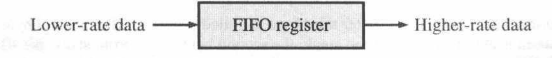


▶ **FIGURE 10–50**
Block diagram of a typical FIFO serial memory.

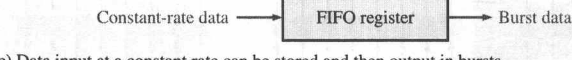
▶ **FIGURE 10–51**
Examples of the FIFO register in data-rate buffering applications.



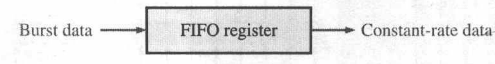
(a) Irregular telemetry data can be stored and retransmitted at a constant rate.



(b) Data input at a slow keyboard rate can be stored and then transferred at a higher rate for processing.



(c) Data input at a constant rate can be stored and then output in bursts.



(d) Data in bursts can be stored and reformatted into a constant-rate output.

Last In–First Out (LIFO) Memories

The **LIFO** (last in–first out) memory is found in applications involving microprocessors and other computing systems. It allows data to be stored and then recalled in reverse order; that is, the last data byte to be stored is the first data byte to be retrieved.

Register Stacks

A LIFO memory is commonly referred to as a push-down stack. In some systems, it is implemented with a group of registers as shown in Figure 10–52. A stack can consist of any number of registers, but the register at the top is called the *top-of-stack*.

To illustrate the principle, a byte of data is loaded in parallel onto the top of the stack. Each successive byte pushes the previous one down into the next register. This process is illustrated in Figure 10–53. Notice that the new data byte is always loaded into the top register and the previously stored bytes are pushed deeper into the stack. The name *push-down stack* comes from this characteristic.

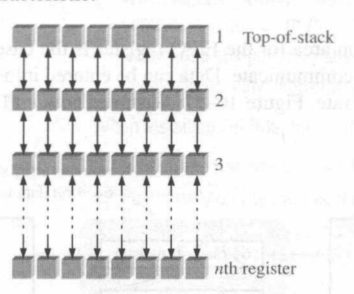


FIGURE 10–52
Register stack.

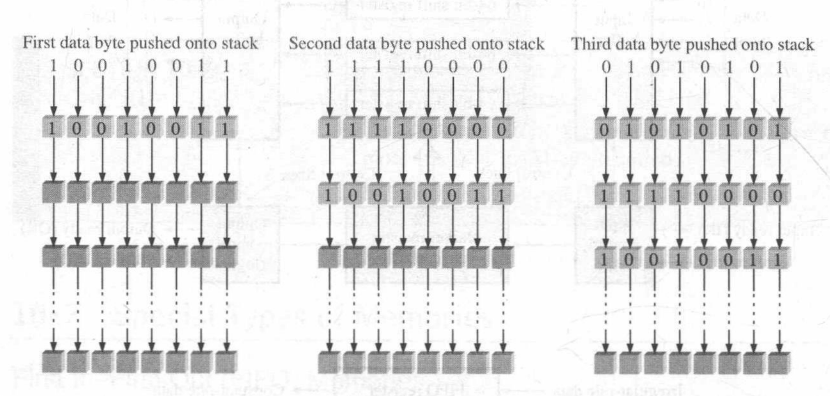


FIGURE 10–53
Simplified illustration of pushing data onto the stack.

Data bytes are retrieved in the reverse order. The last byte entered is always at the top of the stack, so when it is pulled from the stack, the other bytes pop up into the next higher locations. This process is illustrated in Figure 10–54.

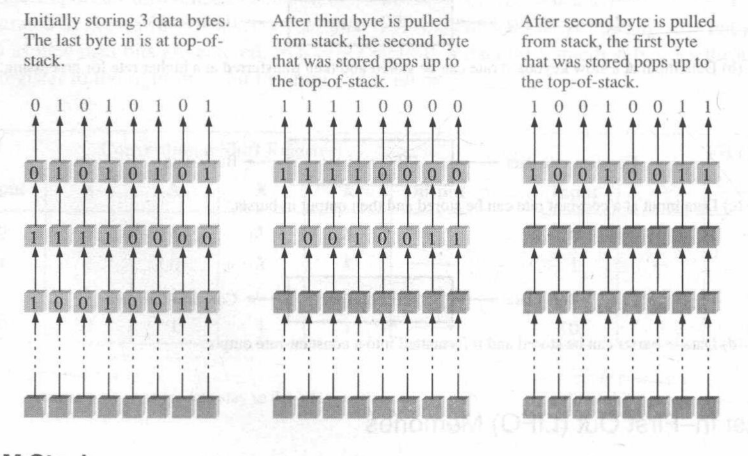
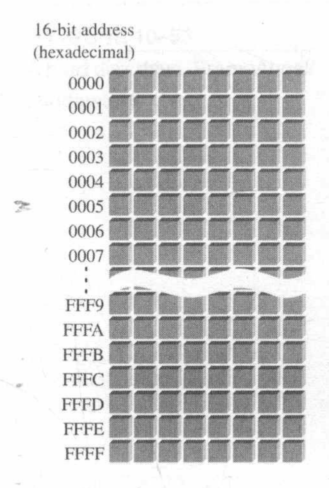


FIGURE 10–54
Simplified illustration of pulling data from the stack.

RAM Stack

Another approach to LIFO memory used in microprocessor-based systems is the allocation of a section of RAM as the stack rather than the use of a dedicated set of registers. As you



▲ **FIGURE 10–55**

Representation of a 64 kB memory with the 16-bit addresses expressed in hexadecimal.

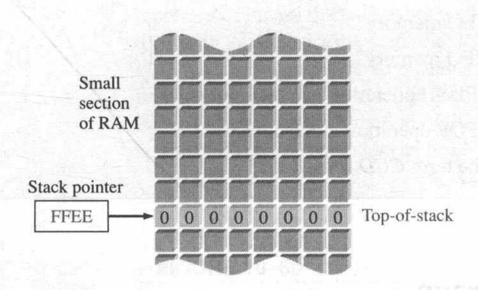
have seen, for a register stack the data move up or down from one location to the next. In a RAM stack, the data do not move but the top-of-stack moves under control of a register called the stack pointer.

Consider a random-access memory that is byte organized—that is, one in which each address contains eight bits—as illustrated in Figure 10–55. The binary address 0000000000001111, for example, can be written as 000F in hexadecimal. A 16-bit address can have a *minimum* hexadecimal value of 0000_{16} and a *maximum* value of $FFFF_{16}$. With this notation, a 64 kB memory array can be represented as shown in Figure 10–55. The lowest memory address is 0000_{16} and the highest memory address is $FFFF_{16}$.

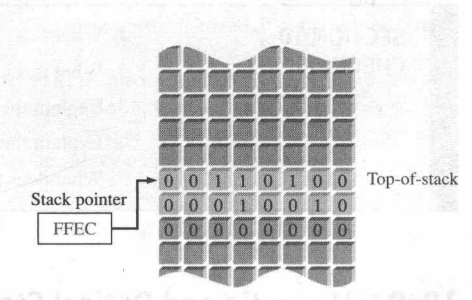
Now, consider a section of RAM set aside for use as a stack. A special separate register, the stack pointer, contains the address of the top of the stack, as illustrated in Figure 10–56. A 4-digit hexadecimal representation is used for the binary addresses. In the figure, the addresses are chosen for purposes of illustration.

Now let's see how data are pushed onto the stack. The stack pointer is initially at address $FFEE_{16}$, which is the top of the stack as shown in Figure 10–56(a). The stack pointer is then decremented (decreased) by two to $FFEC_{16}$. This moves the top of the stack to a lower memory address, as shown in Figure 10–56(b). Notice that the top of the stack is not stationary as in the fixed register stack but moves downward (to lower addresses) in the RAM as data words are stored. Figure 10–56(b) shows that two bytes (one data word) are then pushed onto the stack. After the data word is stored, the top of the stack is at $FFEC_{16}$.

Figure 10–57 illustrates the POP operation for the RAM stack. The last data word stored in the stack is read first. The stack pointer that is at FFEC is incremented (increased) by two to address $FFEE_{16}$ and a POP operation is performed as shown in part (b). Keep in mind that RAMs are nondestructive when read, so the data word still remains in the memory after a POP operation. A data word is destroyed only when a new word is written over it.



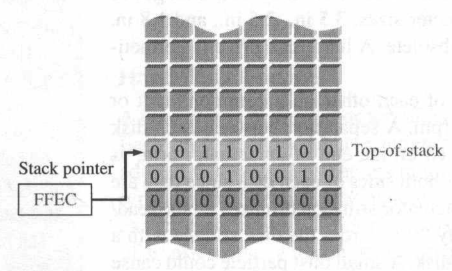
(a) The stack pointer is initially at FFEE before the data word 0001001000110100 (1234) is pushed onto the stack.



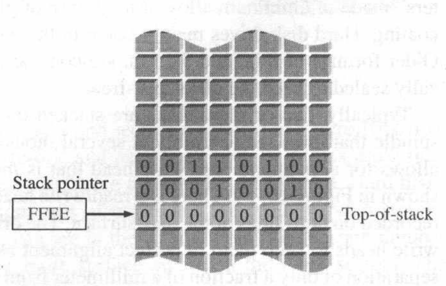
(b) The stack pointer is decremented by two and the data word 0001001000110100 is placed in the two locations prior to the original stack pointer location.

▲ **FIGURE 10–56**

Illustration of the PUSH operation for a RAM stack.



(a) The stack pointer is at FFEC before the data word is copied (popped) from the stack.



(b) The stack pointer is incremented by two and the last data word stored is copied (popped) from the stack.

▲ **FIGURE 10–57**

Illustration of the POP operation for the RAM stack.

A RAM stack can be of any depth, depending on the number of continuous memory addresses assigned for that purpose.

CCD Memories

The **CCD** (charge-coupled device) memory stores data as charges on capacitors and has the ability to convert optical images to electrical signals. Unlike the DRAM, however, the storage cell does not include a transistor. High density is the main advantage of CCDs, and these devices are widely used in digital imaging.

The CCD memory consists of long rows of semiconductor capacitors, called *channels*. Data are entered into a channel serially by depositing a small charge for a 0 and a large charge for a 1 on the capacitors. These charge packets are then shifted along the channel by clock signals as more data are entered.

As with the DRAM, the charges must be refreshed periodically. This process is done by shifting the charge packets serially through a refresh circuit. Figure 10–58 shows the basic concept of a CCD channel. Because data are shifted serially through the channels, the CCD memory has a relatively long access time. CCD arrays are used in many modern cameras to capture video images in the form of light-induced charge.

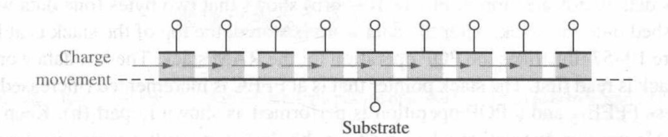


FIGURE 10–58 A CCD (charge-coupled device) channel.

SECTION 10-7 CHECKUP

1. What is a FIFO memory?
2. What is a LIFO memory?
3. Explain the PUSH operation in a memory stack.
4. Explain the POP operation in a memory stack.
5. What does the term *CCD* stand for?

10-8 Magnetic and Optical Storage

Magnetic Storage

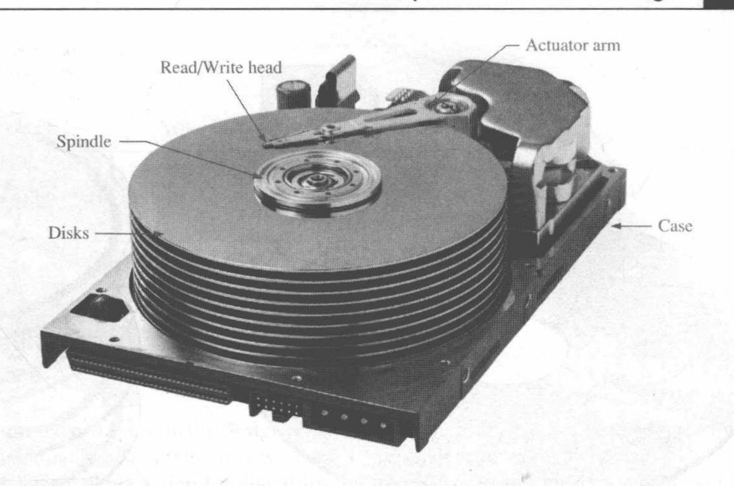
Magnetic Hard Disks

Computers use hard disks as the internal mass storage media. **Hard disks** are rigid "platters" made of aluminum alloy or a mixture of glass and ceramic covered with a magnetic coating. Hard disk drives mainly come in three diameter sizes, 3.5 in., 2.5 in., and 1.8 in. Older formats of 8 in. and 5.25 in. are considered obsolete. A hard disk drive is hermetically sealed to keep the disks dust-free.

Typically, two or more disks are stacked on top of each other on a common shaft or spindle that turns the assembly at several thousand rpm. A separation between each disk allows for a magnetic read/write head that is mounted on the end of an actuator arm, as shown in Figure 10–59. There is a read/write head for both sides of each disk since data are recorded on both sides of the disk surface. The drive actuator arm synchronizes all the read/write heads to keep them in perfect alignment as they "fly" across the disk surface with a separation of only a fraction of a millimeter from the disk. A small dust particle could cause a head to "crash," causing damage to the disk surface.

▶ **FIGURE 10–59**

A hard disk drive. FrameAngel/shutterstock



Basic Read/Write Head Principles

The hard drive is a random-access device because it can retrieve stored data anywhere on the disk in any order. A simplified diagram of the magnetic surface read/write operation is shown in Figure 10–60. The direction or polarization of the magnetic domains on the disk surface is controlled by the direction of the magnetic flux lines (magnetic field) produced by the write head according to the direction of a current pulse in the winding. This magnetic flux magnetizes a small spot on the disk surface in the direction of the magnetic field. A magnetized spot of one polarity represents a binary 1, and one of the opposite polarity represents a binary 0. Once a spot on the disk surface is magnetized, it remains until written over with an opposite magnetic field.

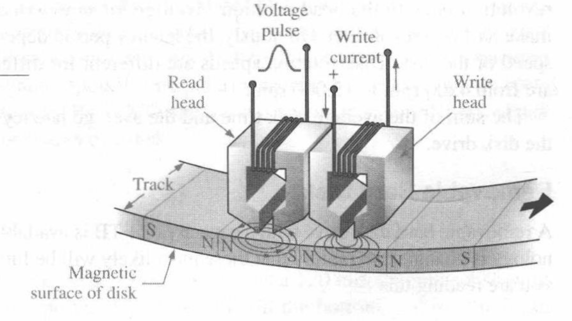
When the magnetic surface passes a read head, the magnetized spots produce magnetic fields in the read head, which induce voltage pulses in the winding. The polarity of these pulses depends on the direction of the magnetized spot and indicates whether the stored bit is a 1 or a 0. The read and write heads are usually combined in a single unit.

InfoNote

Data are stored on a hard drive in the form of files. Keeping track of the location of files is the job of the device driver that manages the hard drive (sometimes referred to as hard drive BIOS). The device driver and the computer's operating system can access two tables to keep track of files and file names. The first table is called the FAT (File Allocation Table). The FAT shows what is assigned to specific files and keeps a record of open sectors and bad sectors. The second table is the Root Directory which has file names, type of file, time and date of creation, starting cluster number, and other information about the file.

▶ **FIGURE 10–60**

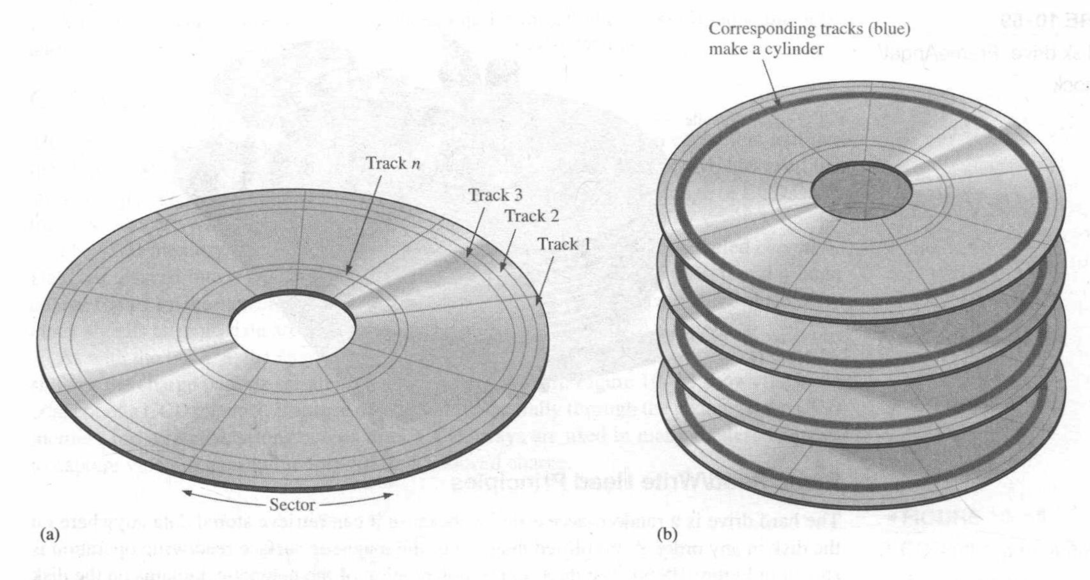
Simplified read/write head operation.



Hard Disk Format

A hard disk is organized or formatted into tracks and sectors, as shown in Figure 10–61(a). Each track is divided into a number of sectors, and each track and sector has a physical address that is used by the operating system to locate a particular data record. Hard disks typically have from a few hundred to thousands of tracks and are available with storage capacities of up to 1 TB or more. As you can see in the figure, there is a constant number of tracks/sector, with outer sectors using more surface area than the inner sectors. The arrangement of tracks and sectors on a disk is known as the *format.*

A hard disk stack is illustrated in Figure 10–61(b). Hard disk drives differ in the number of disks in a stack, but there is always a minimum of two. All of the same corresponding tracks on each disk are collectively known as a cylinder, as indicated.



▲ **FIGURE 10–61**

Hard disk organization and formatting.

Hard Disk Performance

Several basic parameters determine the performance of a given hard disk drive. A *seek* operation is the movement of the read/write head to the desired track. The **seek time** is the average time for this operation to be performed. Typically, hard disk drives have an average seek time of several milliseconds, depending on the particular drive.

The **latency period** is the time it takes for the desired sector to spin under the head once the head is positioned over the desired track. A worst case is when the desired sector is just past the head position and spinning away from it. The sector must rotate almost a full revolution back to the head position. *Average latency period* assumes that the disk must make half of a revolution. Obviously, the latency period depends on the constant rotational speed of the disk. Disk rotation speeds are different for different disk drives but typically are from 4200 rpm to 15,000 rpm.

The sum of the average seek time and the average latency period is the *access time* for the disk drive.

Removable Hard Disk

A removable hard disk drive with a capacity of 1 TB is available. Keep in mind that the technology is changing so rapidly that there most likely will be further advancements at the time you are reading this.

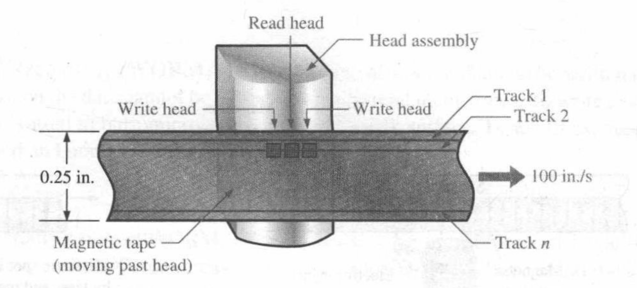
Magnetic Tape

Tape is used for backup data from mass storage devices and typically is slower than disks because data on tape is accessed serially rather than randomly. There are several types that are available, including QIC, 8 mm, and DLT.

QIC is an abbreviation for quarter-inch cartridge and looks much like audio tape cassettes with two reels inside. Various QIC standards have from 28 to 108 tracks that can store from 80 MB to 1.6 GB. More recent innovations under the Travan standard have lengthened the tape and increased its width allowing storage capacities up to 10 GB. QIC tape drives use read/write heads that have a single write head with a read head on each side. This allows the tape drive to verify data just written when the tape is running in either direction. In the record mode, the tape moves past the read/write heads at approximately 100 inches/second, as indicated in Figure 10–62.

InfoNote

Tape is a viable alternative to disk due to its lower cost per bit. Though the density is lower than for a disk drive, the available surface on a tape is far greater. The highest-capacity tape media are generally on the same order as the largest available disk drive (about 1 TB—a terabyte is one trillion bytes.) Tape has historically offered enough advantage in cost over disk storage to make it a viable product, particularly for backup, where media removability is also important.



▲ **FIGURE 10–62**

QIC tape.

8 mm tape was originally designed for the video industry but has been adopted by the computer industry as a reliable way to store large amounts of computer data.

DLT is an abbreviation for digital linear tape. DLT is a half-inch wide tape, which is 60% wider than 8 mm and, of course, twice as wide as standard QIC. Basically, DLT differs in the way the tape-drive mechanism works to minimize tape wear compared to other systems. DLT offers the highest storage capacity of all the tape formats with capacities ranging up to 800 GB.

Magneto-Optical Storage

As the name implies, magneto-optical (MO) storage devices use a combination of magnetic and optical (laser) technologies. A **magneto-optical disk** is formatted into tracks and sectors similar to magnetic disks.

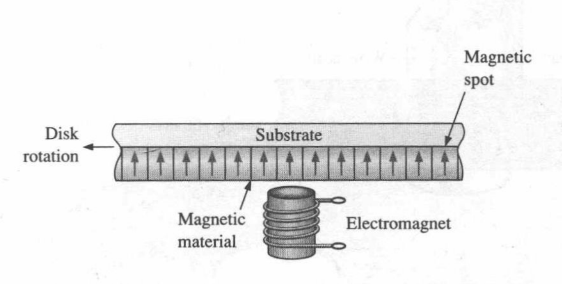
The basic difference between a purely magnetic disk and an MO disk is that the magnetic coating used on the MO disk requires heat to alter the magnetic polarization. Therefore, the MO is extremely stable at ambient temperature, making data unchangeable. To write a data bit, a high-power laser beam is focused on a tiny spot on the disk, and the temperature of that tiny spot is raised above a temperature level called the Curie point (about 200°C). Once heated, the magnetic particles at that spot can easily have their direction (polarization) changed by a magnetic field generated by the write head. Information is read from the disk with a less-powerful laser than used for writing, making use of the Kerr effect where the polarity of the reflected laser light is altered depending on the orientation of the magnetic particles. Magnetic spots of one polarity represent 0s and magnetic spots of the opposite polarity represent 1s. Basic MO operation is shown in Figure 10–63, which represents a small cross-sectional area of a disk.

Optical Storage

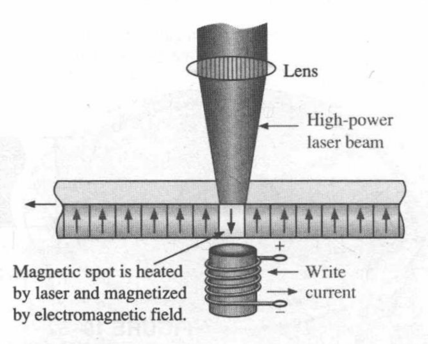
CD-ROM

The most common Compact Disk–Read-Only Memory is a 120 mm diameter disk with a sandwich of three coatings: a polycarbonate plastic on the bottom, a thin aluminum sheet for reflectivity, and a top coating of lacquer for protection. The **CD-ROM** disk is formatted in a single spiral track with sequential 2 kB sectors and has a capacity of 680 MB. Data are prerecorded at the factory in the form of minute indentations called *pits* and the flat area surrounding the pits called *lands*. The pits are stamped into the plastic layer and cannot be erased.

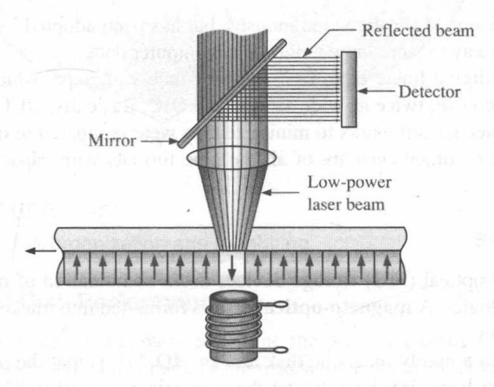
A CD player reads data from the spiral track with a low-power infrared laser, as illustrated in Figure 10–64. The data are in the form of pits and lands as shown. Laser light reflected from a pit is 180° out-of-phase with the light reflected from the lands. As the disk rotates, the narrow laser beam strikes the series of pits and lands of varying lengths, and a photodiode detects the difference in the reflected light. The result is a series of 1s and 0s corresponding to the configuration of pits and lands along the track.



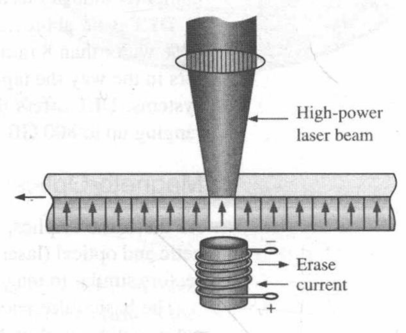
(a) Unrecorded disk



(b) Writing: A high-power laser beam heats the spot, causing the magnetic particles to align with the electromagnetic field.



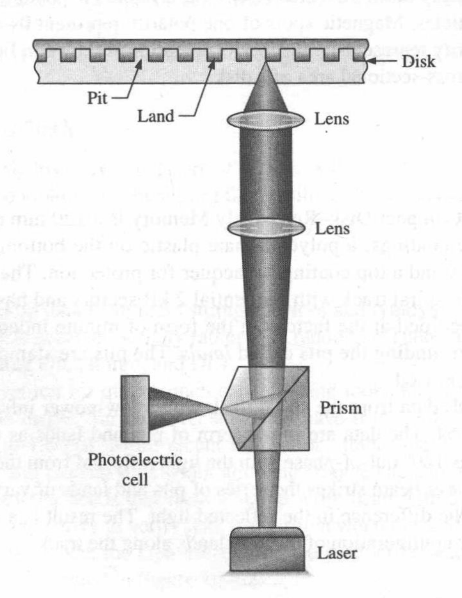
(c) Reading: A low-power laser beam reflects off of the reversed-polarity magnetic particles and its polarization shifts. If the particles are not reversed, the polarization of the reflected beam is unchanged.



(d) Erasing: The electromagnetic field is reversed as the high-power laser beam heats the spot, causing the magnetic particles to be restored to the original polarity.

▲FIGURE 10–63

Basic principle of a magneto-optical disk.



◄FIGURE 10–64

Basic operation of reading data from a CD-ROM.

WORM

Write Once/Read Many (**WORM**) is a type of optical storage that can be written onto one time after which the data cannot be erased but can be read many times. To write data, a low-power laser is used to burn microscopic pits on the disk surface. 1s and 0s are represented by the burned and nonburned areas.

CD-R

This is essentially a type of WORM. The difference is that the CD-Recordable allows multiple write sessions to different areas of the disk. The **CD-R** disk has a spiral track like the CD-ROM, but instead of mechanically pressing indentations on the disk to represent data, the CD-R uses a laser to burn microscopic spots into an organic dye surface. When heated beyond a critical temperature with a laser during read, the burned spots change color and reflect less light than the nonburned areas. Therefore, 1s and 0s are represented on a CD-R by burned and nonburned areas, whereas on a CD-ROM they are represented by pits and lands. Like the CD-ROM, the data cannot be erased once it is written.

CD-RW

The CD-Rewritable disk can be used to read and write data. Instead of the dye-based recording layer in the CD-R, the **CD-RW** commonly uses a crystalline compound with a special property. When it is heated to a certain temperature, it becomes crystalline when it cools; but if it is heated to a certain higher temperature, it melts and becomes amorphous when it cools. To write data, the focused laser beam heats the material to the melting temperature resulting in an amorphous state. The resulting amorphous areas reflect less light than the crystalline areas, allowing the read operation to detect 1s and 0s. The data can be erased or overwritten by heating the amorphous areas to a temperature above the crystallization temperature but lower than the melting temperature that causes the amorphous material to revert to a crystalline state.

DVD-ROM

Originally DVD was an abbreviation for Digital Video Disk but eventually came to represent *Digital Versatile Disk.* Like the CD-ROM, **DVD-ROM** data are prestored on the disk. However, the pit size is smaller than for the CD-ROM, allowing more data to be stored on a track. The major difference between CD-ROM and DVD-ROM is that the CD is single-sided, while the DVD has data on both sides. Also, in addition to double-sided DVD disks, there are also multiple-layer disks that use semitransparent data layers placed over the main data layers, providing storage capacities of tens of gigabytes. To access all the layers, the laser beam requires refocusing going from one layer to the other.

Blu-Ray

The **Blu-ray** Disc (BD) is designed to eventually replace the DVD. The BD is the same size as DVDs and CDs. The name *Blu-ray* refers to the blue laser used to read the disc. DVDs use a red laser that has a longer wavelength. Information can be stored on a BD at a greater density and video definition than is possible with a DVD. The smaller Blu-ray laser beam can read recorded data in pits that are less than half the size of the pits on a DVD. A Blu-ray Disc can store about five times more data than a DVD. Typical storage capacities for conventional Blu-ray dual-layer discs are 50 GB, which is the industry standard for feature-length video. Triple layer and quadruple layer discs (BD-XL) can store 100 GB and 128 GB, respectively. Storage capacities up to 1 TB are currently under development.

SECTION 10-8 CHECKUP

1. List the major types of magnetic storage.
2. Generally, how is a magnetic disk organized?
3. How are data written on and read from a magneto-optical disk?
4. List the types of optical storage.

TRUE/FALSE QUIZ

Answers are at the end of the chapter.

1. A data byte consists of eight bits.
2. A memory cell can store a byte of data.
3. The write operation stores data in memory.
4. The read operation always erases the data byte.
5. RAM is a *random address memory*.
6. Stored data are lost if power is removed from a static RAM.
7. Cache is a type of memory used for intermediate or temporary storage of data.
8. Dynamic RAMs must be periodically refreshed to retain data.
9. ROM is a *random output memory*.
10. A flash memory uses a flashing beam of light to store data.

SELF-TEST

Answers are at the end of the chapter.

1. The bit capacity of a memory that has 1024 addresses and can store 8 bits at each address is
   (a) 1024 (b) 8192 (c) 8 (d) 4096
2. A 32-bit data word consists of
   (a) 2 bytes (b) 4 nibbles (c) 4 bytes (d) 3 bytes and 1 nibble
3. Data are stored in a random-access memory (RAM) during the
   (a) read operation (b) enable operation
   (c) write operation (d) addressing operation
4. Data that are stored at a given address in a random-access memory (RAM) are lost when
   (a) power goes off (b) the data are read from the address
   (c) new data are written at the address (d) answers (a) and (c)
5. A ROM is a
   (a) nonvolatile memory (b) volatile memory
   (c) read/write memory (d) byte-organized memory
6. A memory with 256 addresses has
   (a) 256 address lines (b) 6 address lines
   (c) 1 address line (d) 8 address lines
7. A byte-organized memory has
   (a) 1 data output line (b) 4 data output lines
   (c) 8 data output lines (d) 16 data output lines
8. The storage cell in a SRAM is
   (a) a flip-flop (b) a capacitor (c) a fuse (d) a magnetic domain
9. A DRAM must be
   (a) replaced periodically (b) refreshed periodically
   (c) always enabled (d) programmed before each use
10. A flash memory is
    (a) volatile (b) a read-only memory
    (c) a read/write memory (d) nonvolatile
    (e) answers (a) and (c) (f) answers (c) and (d)
11. SRAM, DRAM, flash, and EEPROM are all
    (a) magneto-optical storage devices (b) semiconductor storage devices
    (c) magnetic storage devices (d) optical storage devices
12. Optical storage devices employ
    (a) ultraviolet light (b) electromagnetic fields
    (c) optical couplers (d) lasers

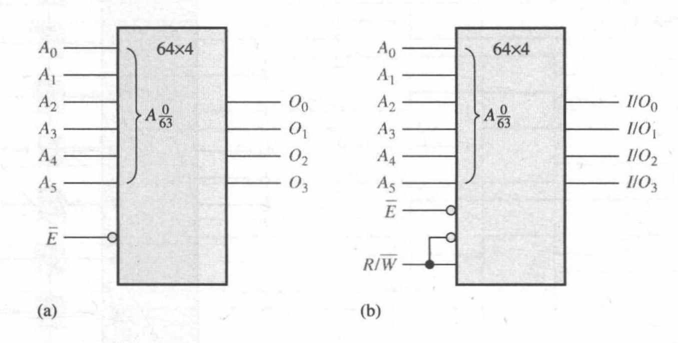
PROBLEMS

Answers to odd-numbered problems are at the end of the book.

Section 10–1 Semiconductor Memory Basics

1. Identify the ROM and the RAM in Figure 10–65.

FIGURE 10–65



2. Explain why RAMs and ROMs are both random-access memories.
3. Explain the purposes of the address bus and the data bus.
4. What memory address (0 through 256) is represented by each of the following hexadecimal numbers:

   (a) $0A_{16}$ (b) $3F_{16}$ (c) CD_{16}

Section 10–2 The Random-Access Memory (RAM)

5. A static memory array with four rows similar to the one in Figure 11–10 is initially storing all 0s. What is its content after the following conditions? Assume a 1 selects a row.

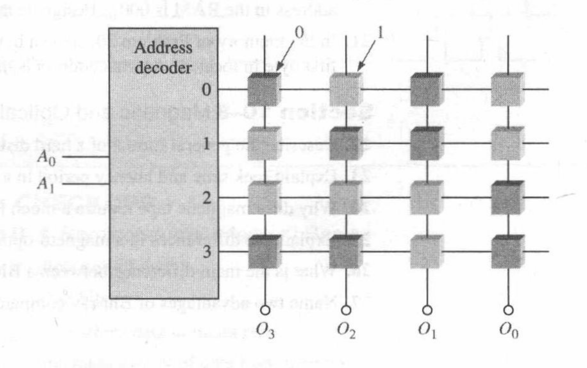
   Row 0 = 1, Data in (Bit 0) = 1
   Row 1 = 0, Data in (Bit 1) = 1
   Row 2 = 1, Data in (Bit 2) = 1
   Row 3 = 0, Data in (Bit 3) = 0

6. Draw a basic logic diagram for a 512 × 8-bit static RAM, showing all the inputs and outputs.
7. Assuming that a 64k × 8 SRAM has a structure similar to that of the SRAM in Figure 10–12. determine the number of rows and 8-bit columns in its memory cell array.
8. Redraw the block diagram in Figure 10–12 for a 64k × 8 memory.
9. Explain the difference between a SRAM and a DRAM.
10. What is the capacity of a DRAM that has twelve address lines?

Section 10–3 The Read-Only Memory (ROM)

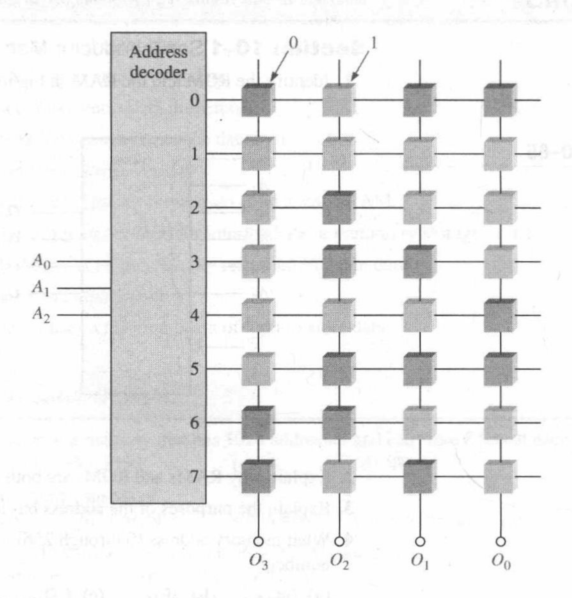
11. For the ROM array in Figure 10–66, determine the outputs for all possible input combinations, and summarize them in tabular form (Blue cell is a 1, gray cell is a 0).

FIGURE 10–66



12. Determine the truth table for the ROM in Figure 10–67.

► **FIGURE 10–67**



13. Using a procedure similar to that in Example 10–1, design a ROM for conversion of single-digit BCD to excess-3 code.
14. What is the total *bit* capacity of a ROM that has 14 address lines and 8 data outputs?

Section 10–4 Programmable ROMs

15. Determine the addresses that are programmed and the contents of each address after the programming sequence in Figure 10–68 has been applied to an EPROM like the one shown in Figure 10–31.

Section 10–6 Memory Expansion

16. Use 16k × 4 DRAMs to build a 64k × 8 DRAM. Show the logic diagram.
17. Using a block diagram, show how 64k × 1 dynamic RAMs can be expanded to build a 256k × 4 RAM.
18. What is the word length and the word capacity of the memory of Problem 16? Problem 17?

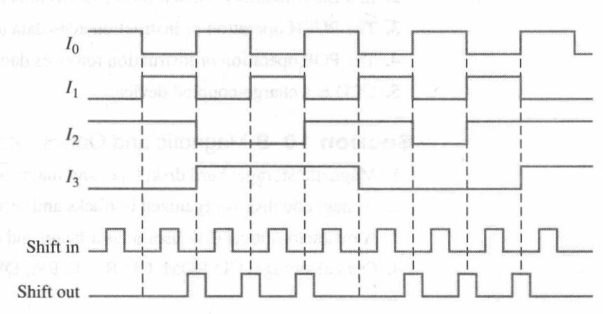
Section 10–7 Special Types of Memories

19. Complete the timing diagram in Figure 10–69 by showing the output waveforms that are initially all LOW for a FIFO serial memory like that shown in Figure 10–50.
20. Consider a 4096 × 8 RAM in which the last 64 addresses are used as a LIFO stack. If the first address in the RAM is 000_{16}, designate the 64 addresses used for the stack.
21. In the memory of Problem 20, sixteen bytes are pushed into the stack. At what address is the first byte in located? At what address is the last byte in located?

Section 10–8 Magnetic and Optical Storage

22. Describe the general format of a hard disk.
23. Explain seek time and latency period in a hard disk drive.
24. Why does magnetic tape require a much longer access time than does a disk?
25. Explain the differences in a magneto-optical disk, a CD-ROM, and a WORM.
26. What is the main difference between a Blu-ray Disc and a DVD?
27. Name two advantages of Blu-ray compared to a DVD.

▶ FIGURE 10–68



▶ FIGURE 10–69

ANSWERS

SECTION CHECKUPS

Section 10–1 Semiconductor Memory Basics

1. Bit is the smallest unit of data.
2. 256 bytes is 2048 bits.
3. A write operation stores data in memory.
4. A read operation takes a copy of data from memory.
5. A unit of data is located by its address.
6. A RAM is volatile and has read/write capability. A ROM is nonvolatile and has only read capability.

Section 10–2 The Random-Access Memory (RAM)

1. Asynchronous and synchronous with burst feature
2. A small fast memory between the CPU and main memory
3. SRAMs have latch storage cells that can retain data indefinitely while power is applied. DRAMs have capacitive storage cells that must be periodically refreshed.
4. The refresh operation prevents data from being lost because of capacitive discharge. A stored bit is restored periodically by recharging the capacitor to its nominal level.
5. FPM, EDO, BEDO, Synchronous

Section 10–3 The Read-Only Memory (ROM)

1. 512×8 equals 4096 bits.
2. Mask ROM, PROM, EPROM, UV EPROM, EEPROM
3. Eight bits of address are required for 256 byte locations ($2^8 = 256$).

Section 10–4 Programmable ROMs

1. PROMs are field-programmable; ROMs are not.
2. Presence or absence of stored charge
3. Read is the normal mode of operation for a PROM.

Section 10–5 The Flash Memory

1. Flash, ROM, EPROM, and EEPROM are nonvolatile.
2. Flash is nonvolatile; SRAM and DRAM are volatile.
3. Programming, read, erase

Section 10–6 Memory Expansion

1. Eight RAMs
2. Eight RAMs
3. DIMM: Dual in-line memory module

Section 10–7 Special Types of Memories

1. In a FIFO memory the *first* bit (or word) *in* is the *first* bit (or word) *out.*
2. In a LIFO memory the *last* bit (or word) *in* is the *first* bit (or word) *out.* A stack is a LIFO.
3. The PUSH operation or instruction adds data to the memory stack.
4. The POP operation or instruction removes data from the memory stack.
5. CCD is a charge-coupled device.

Section 10–8 Magnetic and Optical Storage

1. Magnetic storage: hard disk, tape, and magneto-optical disk
2. A magnetic disk is organized in tracks and sectors.
3. A magneto-optical disk uses a laser beam and an electromagnet.
4. Optical storage: CD-ROM, CD-R, CD-RW, DVD-ROM, WORM, Blu-ray Disc (BD)

RELATED PROBLEMS FOR EXAMPLES

10–1 $G_3G_2G_1G_0 = 1110$

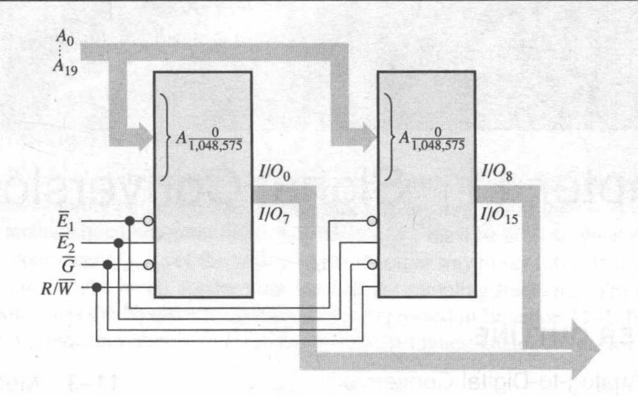
10–2 Connect eight $64\text{k} \times 1$ ROMs in parallel to form a $64\text{k} \times 8$ ROM.

10–3 Sixteen $64\text{k} \times 1$ ROMs

10–4 See Figure 10–70.

10–5 ROM 1: 0 to 524,287; ROM 2: 524,288 to 1,048,575

▶ **FIGURE 10–70**



TRUE/FALSE QUIZ

1. T **2.** F **3.** T **4.** F **5.** F **6.** T
7. T **8.** T **9.** F **10.** F

SELF-TEST

1. (b) **2.** (c) **3.** (c) **4.** (d) **5.** (a) **6.** (d) **7.** (c)
8. (a) **9.** (b) **10.** (f) **11.** (b) **12.** (d)

Chapter 11 Signal Conversion and Processing

CHAPTER OUTLINE

11-1 Analog-to-Digital Conversion

Sampling and Filtering

An anti-aliasing filter and a sample-and-hold circuit are two functions typically found in a digital signal processing system. The sample-and-hold function does two operations, the first of which is sampling. **Sampling** is the process of taking a sufficient number of discrete values at points on a waveform that will define the shape of the waveform. The more samples you take, the more accurately you can define a waveform. Sampling converts an analog signal into a series of impulses, each representing the amplitude of the signal at a given instant in time. Figure 11–1 illustrates the process of sampling.

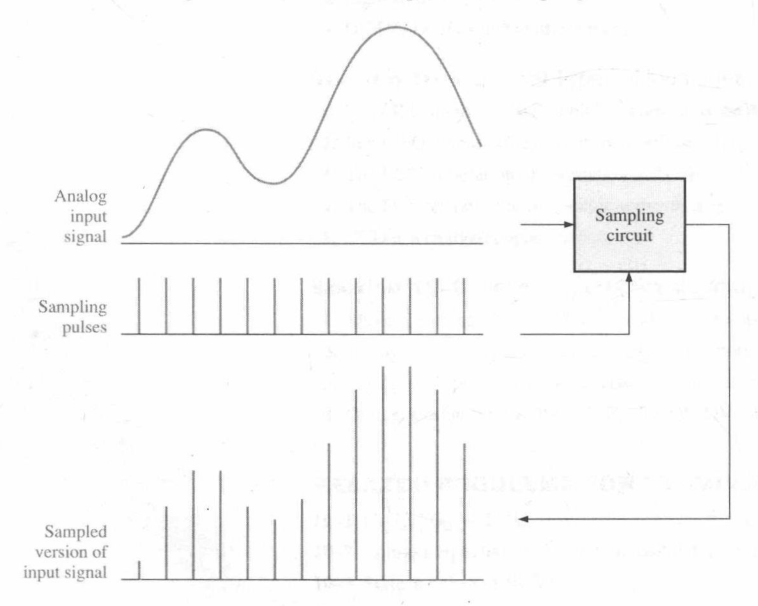


FIGURE 11–1
Illustration of the sampling process.

When an analog signal is to be sampled, there are certain criteria that must be met in order to accurately represent the original signal. All analog signals (except a pure sine wave) contain a spectrum of component frequencies. For a pure sine wave, these frequencies appear in multiples called *harmonics*. The harmonics of an analog signal are sine waves of different frequencies and amplitudes. When the harmonics of a given periodic

waveform are added, the result is the original signal. Before a signal can be sampled, it must be passed through a low-pass filter (anti-aliasing filter) to eliminate harmonic frequencies above a certain value as determined by the Nyquist frequency.

The Sampling Theorem

Notice in Figure 11–1 that there are two input waveforms. One is the analog signal and the other is the sampling pulse waveform. The sampling theorem states that, in order to represent an analog signal, the sampling frequency, f_{sample}, must be at least twice the highest frequency component $f_{a(max)}$ of the analog signal. Another way to say this is that the highest analog frequency can be no greater than one-half the sampling frequency. The frequency $f_{a(max)}$ is known as the **Nyquist frequency** and is expressed in Equation 11–1. In practice, the sampling frequency should be more than twice the highest analog frequency.

$$f_{sample} > 2f_{a(max)} \qquad \text{Equation 11–1}$$

To intuitively understand the sampling theorem, a simple "bouncing-ball" analogy may be helpful. Although it is not a perfect representation of the sampling of electrical signals, it does serve to illustrate the basic idea. If a ball is photographed (sampled) at one instant during a single bounce, as illustrated in Figure 11–2(a), you cannot tell anything about the path of the ball except that it is off the floor. You can't tell whether it is going up or down or the distance of its bounce. If you take photos at two equally-spaced instants during one bounce, as shown in part (b), you can obtain only a minimum amount of information about its movement and nothing about the distance of the bounce. In this particular case, you know only that the ball has been in the air at the times the two photos were taken and that the maximum height of the bounce is at least equal to the height shown in each photo. If you take four photos, as shown in part (c), then the path that the ball follows during a bounce begins to emerge. The more photos (samples) that you take, the more accurately you can determine the path of the ball as it bounces.

(a) One sample of a ball during a single bounce

(b) Two samples of a ball during a single bounce. This is the absolute minimum required to tell anything about its movement, but generally insufficient to describe its path.

(c) Four samples of a ball during a single bounce form a rough picture of the path of the ball.

▲ **FIGURE 11–2**

Bouncing ball analogy of sampling theory.

The Need for Filtering

Low-pass filtering is necessary to remove all frequency components (harmonics) of the analog signal that exceed the Nyquist frequency. If there are any frequency components in the analog signal that exceed the Nyquist frequency, an unwanted condition known as **aliasing** will occur. An alias is a signal produced when the sampling frequency is not at least twice the signal frequency. An alias signal has a frequency that is less than the highest frequency in the analog signal being sampled and therefore falls within the spectrum or frequency band of the input analog signal causing distortion. Such a signal is actually "posing" as part of the analog signal when it really isn't, thus the term *alias.*

Another way to view aliasing is by considering that the sampling pulses produce a spectrum of harmonic frequencies above and below the sample frequency, as shown in Figure 11–3. If the analog signal contains frequencies above the Nyquist frequency, these frequencies overlap into the spectrum of the sample waveform as shown and interference occurs. The lower frequency components of the sampling waveform become mixed in with the frequency spectra of the analog waveform, resulting in an aliasing error.

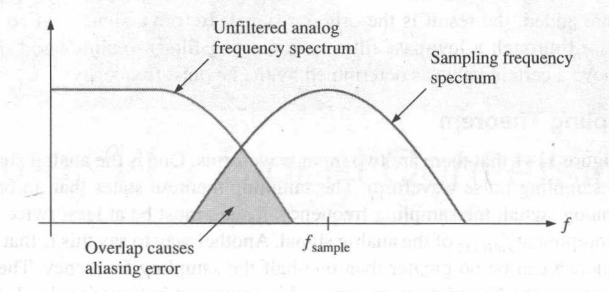


FIGURE 11–3 A basic illustration of the condition $f_{sample} < 2f_{a(max)}$.

A low-pass anti-aliasing filter must be used to limit the frequency spectrum of the analog signal for a given sample frequency. To avoid an aliasing error, the filter must at least eliminate all analog frequencies above the minimum frequency in the sampling spectrum, as illustrated in Figure 11–4. Aliasing can also be avoided by sufficiently increasing the sampling frequency. However, the maximum sampling frequency is usually limited by the performance of the analog-to-digital converter (ADC) that follows it.

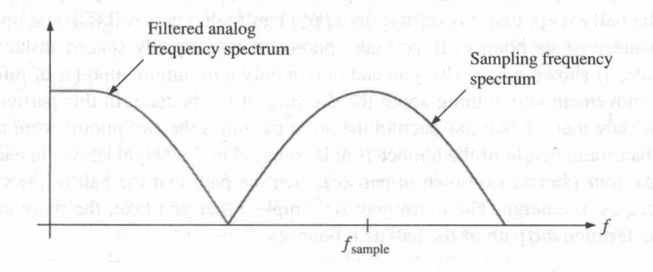


FIGURE 11–4 After low-pass filtering, the frequency spectra of the analog and the sampling signals do not overlap, thus eliminating aliasing error.

An Application

An example of the application of sampling is in digital audio equipment. The sampling rates used are 32 kHz, 44.1 kHz, or 48 kHz (the number of samples per second). The 48 kHz rate is the most common, but the 44.1 kHz rate is used for audio CDs and prerecorded tapes. According to the Nyquist rate, the sampling frequency must be at least twice the audio signal. Therefore, the CD sampling rate of 44.1 kHz captures frequencies up to about 22 kHz, which exceeds the 20 kHz specification that is common for most audio equipment.

Many applications do not require a wide frequency range to obtain reproduced sound that is acceptable. For example, human speech contains some frequencies near 10 kHz and, therefore, requires a sampling rate of at least 20 kHz. However, if only frequencies up to 4 kHz (ideally requiring an 8 kHz minimum sampling rate) are reproduced, voice is very understandable. On the other hand, if a sound signal is not sampled at a high enough rate, the effect of aliasing will become noticeable with background noise and distortion.

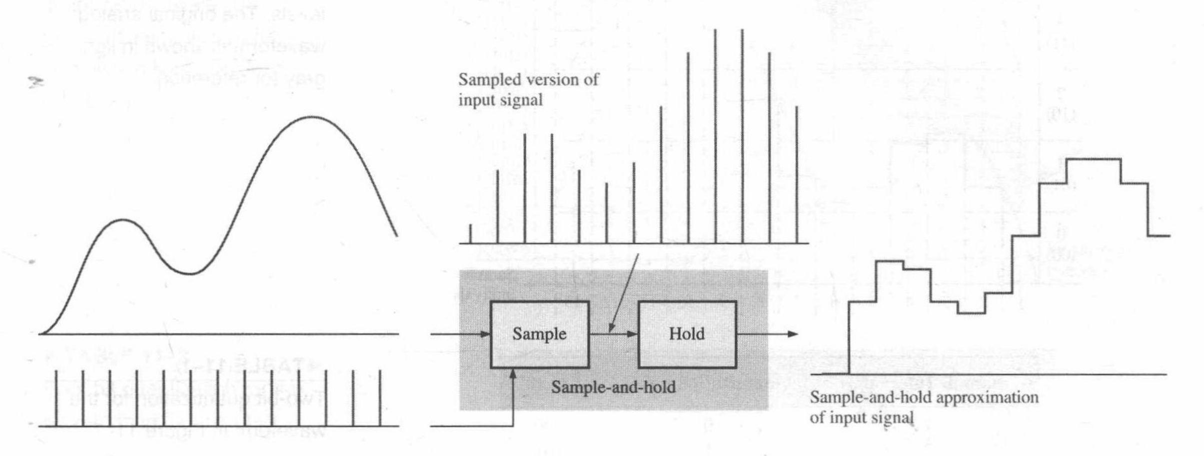
Holding the Sampled Value

The holding operation is the second part of the sample-and-hold function. After filtering and sampling, the sampled level must be held constant until the next sample occurs. This is necessary for the ADC to have time to process the sampled value. This sample-and-hold operation results in a "stairstep" waveform that approximates the analog input waveform, as shown in Figure 11–5.

Analog-to-Digital Conversion

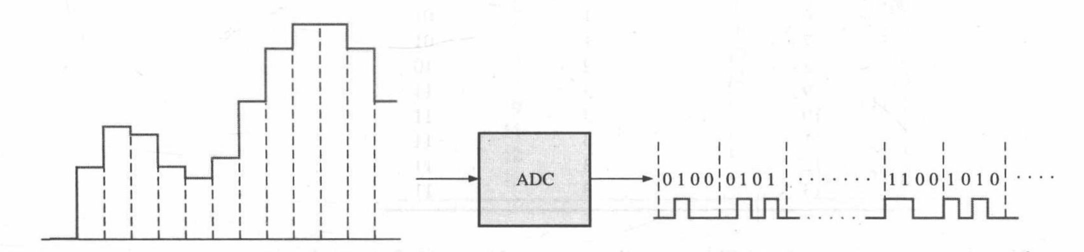
Analog-to-digital conversion is the process of converting the output of the sample-and-hold circuit to a series of binary codes that represent the amplitude of the analog input at each of the sample times. The sample-and-hold process keeps the amplitude of the analog input signal constant between sample pulses; therefore, the analog-to-digital conversion can be done using a constant value rather than having the analog signal change during a conversion interval, which is the time between sample pulses.

Figure 11–6 illustrates the basic function of an **analog-to-digital converter (ADC)**, which is a circuit that performs analog-to-digital conversion. The sample intervals are indicated by dashed lines.



▲ **FIGURE 11–5**

Illustration of a sample-and-hold operation.



▲ **FIGURE 11–6**

Basic function of an analog-to-digital converter (ADC) (The binary codes and number of bits are arbitrarily chosen for illustration only). The ADC output waveform that represents the binary codes is also shown.

Quantization

The process of converting an analog value to a code is called **quantization.** During the quantization process, the ADC converts each sampled value of the analog signal to a binary code. The more bits that are used to represent a sampled value, the more accurate is the representation.

To illustrate, let's quantize a reproduction of the analog waveform into four levels (0–3). Two bits are required for four levels. As shown in Figure 11–7, each quantization level is represented by a 2-bit code on the vertical axis, and each sample interval is numbered along the horizontal axis. The sampled data is held for the entire sample period. This data is quantized to the next lower level, as shown in Table 11–1 (for example, compare samples 3 and 4, which are assigned different levels).

If the resulting 2-bit digital codes are used to reconstruct the original waveform, you would get the waveform shown in Figure 11–8. This operation is done by **digital-to-analog converters (DACs)**, which are circuits that perform digital-to-analog conversions. As you can see, quite a bit of accuracy is lost using only two bits to represent the sampled values.

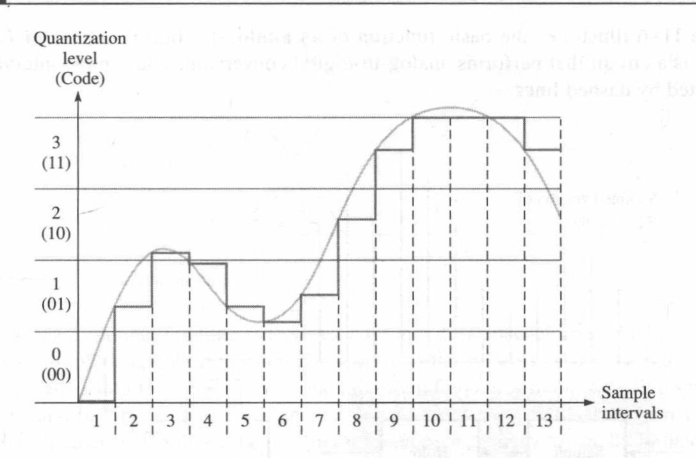


FIGURE 11–7
Sample-and-hold output waveform with four quantization levels. The original analog waveform is shown in light gray for reference.

TABLE 11–1
Two-bit quantization for the waveform in Figure 11–7.

Sample Interval	Quantization Level	Code
1	0	00
2	1	01
3	2	10
4	1	01
5	1	01
6	1	01
7	1	01
8	2	10
9	3	11
10	3	11
11	3	11
12	3	11
13	3	11

Now, let's see how more bits will improve the accuracy. Figure 11–9 shows the same waveform with sixteen quantization levels (4 bits). The 4-bit quantization process is summarized in Table 11–2.

If the resulting 4-bit digital codes are used to reconstruct the original waveform, you would get the waveform shown in Figure 11–10. As you can see, the result is much more like the original waveform than for the case of four quantization levels in Figure 11–8. This shows that greater accuracy is achieved with more quantization bits. Typical integrated circuit ADCs use from 12 to 24 bits, and the sample-and-hold function is sometimes contained on the ADC chip. Several types of ADCs are introduced in the next section.

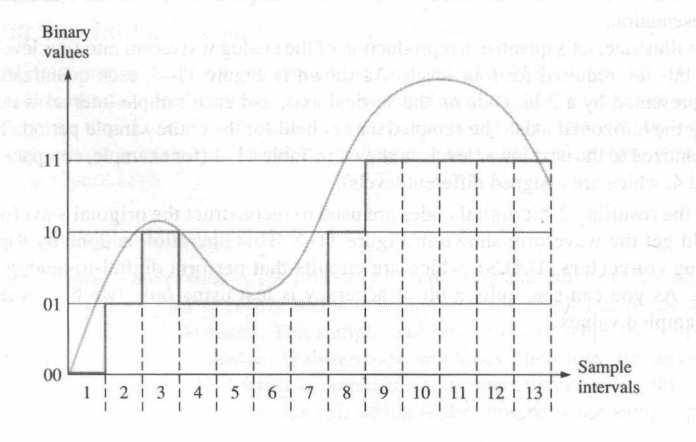


FIGURE 11–8
The reconstructed waveform in Figure 11–7 using four quantization levels (2 bits). The original analog waveform is shown in light gray for reference.

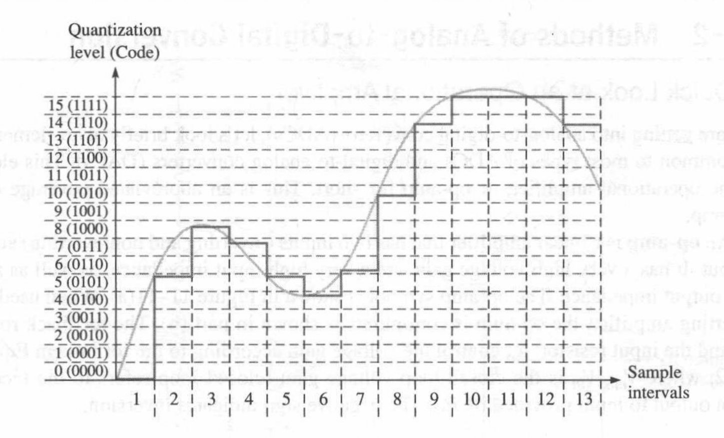


FIGURE 11–9 Sample-and-hold output waveform with sixteen quantization levels. The original analog waveform is shown in light gray for reference.

TABLE 11–2 Four-bit quantization for the waveform in Figure 11–9.

Sample Interval	Quantization Level	Code
1	0	0000
2	5	0101
3	8	1000
4	7	0111
5	5	0101
6	4	0100
7	6	0110
8	10	1010
9	14	1110
10	15	1111
11	15	1111
12	15	1111
13	14	1110

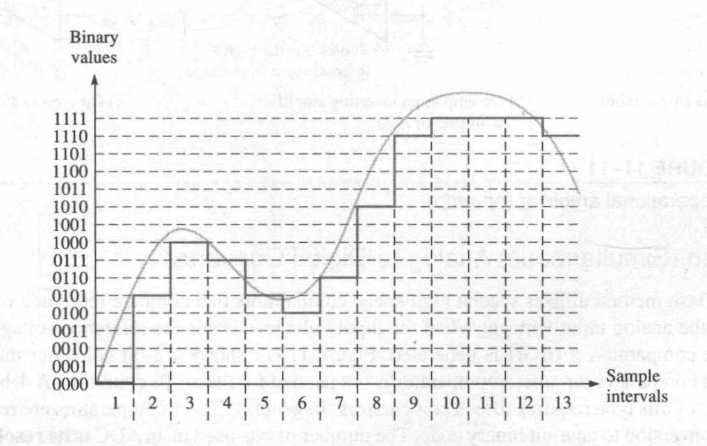


FIGURE 11–10 The reconstructed waveform in Figure 11–9 using sixteen quantization levels (4 bits). The original analog waveform is shown in light gray for reference.

SECTION 11-1 CHECKUP

Answers are at the end of the chapter.

1. What does sampling mean?
2. Why must you hold a sampled value?
3. If the highest frequency component in an analog signal is 20 kHz, what is the minimum sample frequency?
4. What does quantization mean?
5. What determines the accuracy of the quantization process?

11-2 Methods of Analog-to-Digital Conversion

A Quick Look at an Operational Amplifier

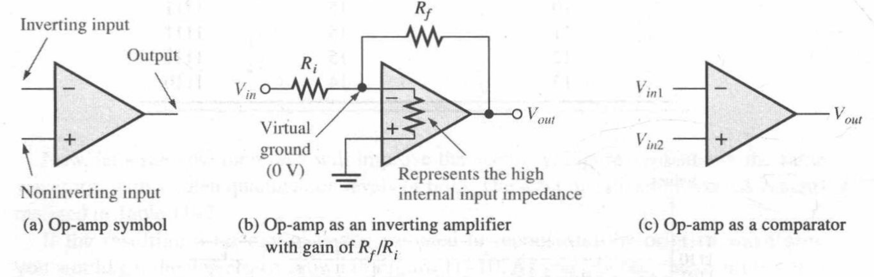
Before getting into analog-to-digital converters (ADCs), let's look briefly at an element that is common to most types of ADCs and digital-to-analog converters (DACs). This element is the operational amplifier, or op-amp for short. This is an abbreviated coverage of the op-amp.

An **op-amp** is a linear amplifier that has two inputs (inverting and noninverting) and one output. It has a very high voltage gain and a very high input impedance, as well as a very low output impedance. The op-amp symbol is shown in Figure 11–11(a). When used as an inverting amplifier, the op-amp is configured as shown in part (b). The feedback resistor, R_f, and the input resistor, R_i, control the voltage gain according to the formula in Equation 11–2, where V_{out}/V_{in} is the closed-loop voltage gain (closed loop refers to the feedback from output to input provided by R_f). The negative sign indicates inversion.

$$\frac{V_{out}}{V_{in}} = -\frac{R_f}{R_i} \quad \text{Equation 11–2}$$

In the inverting amplifier configuration, the inverting input of the op-amp is approximately at ground potential (0 V) because feedback and the extremely high open-loop gain make the differential voltage between the two inputs extremely small. Since the noninverting input is grounded, the inverting input is at approximately 0 V, which is called *virtual ground.*

When the op-amp is used as a comparator, as shown in Figure 11–11(c), two voltages are applied to the inputs. When these input voltages differ by a very small amount, the op-amp is driven into one of its two saturated output states, either HIGH or LOW, depending on which input voltage is greater.



▲ **FIGURE 11–11**
The operational amplifier (op-amp).

Flash (Simultaneous) Analog-to-Digital Converter

The flash method utilizes special high-speed comparators that compare reference voltages with the analog input voltage. When the input voltage exceeds the reference voltage for a given comparator, a HIGH is generated. Figure 11–12 shows a 3-bit converter that uses seven comparator circuits; a comparator is not needed for the all-0s condition. A 4-bit converter of this type requires fifteen comparators. In general, $2^n - 1$ comparators are required for conversion to an n-bit binary code. The number of bits used in an ADC is its **resolution**. The large number of comparators necessary for a reasonable-sized binary number is one of the disadvantages of the **flash ADC**. Its chief advantage is that it provides a fast conversion time because of a high *throughput*, measured in samples per second (sps).

The reference voltage for each comparator is set by the resistive voltage-divider circuit. The output of each comparator is connected to an input of the priority encoder. The encoder is enabled by a pulse on the *EN* input, and a 3-bit code representing the value of the input appears on the encoder's outputs. The binary code is determined by the highest-order input having a HIGH level.

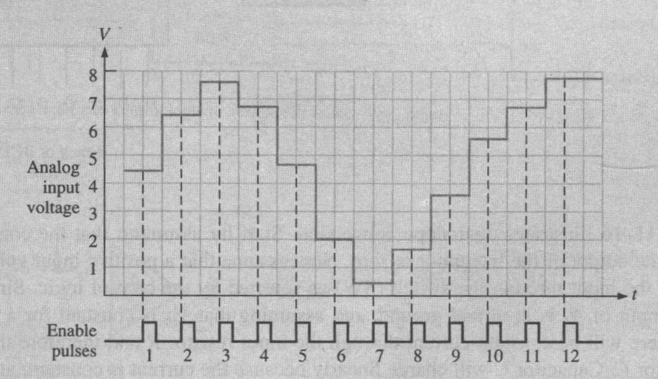
The frequency of the enable pulses and the number of bits in the binary code determine the accuracy with which the sequence of binary codes represents the input of the ADC. The signal is sampled each time the enable pulse is active.

▲ **FIGURE 11–12**

A 3-bit flash ADC.

EXAMPLE 11-1

Determine the binary code output of the 3-bit flash ADC in Figure 11–12 for the input signal in Figure 11–13 and the encoder enable pulses shown. For this example, $V_{REF} = +8$ V.



▶ **FIGURE 11–13**

Sampling of values on a waveform for conversion to binary code.

Solution

The resulting digital output sequence is listed as follows and shown in the waveform diagram of Figure 11–14 in relation to the enable pulses:

100, 110, 111, 110, 100, 010, 000, 001, 011, 101, 110, 111

Related Problem*

If the enable pulse frequency in Figure 11–13 were halved, determine the binary numbers represented by the resulting digital output sequence for 6 pulses. Is any information lost?

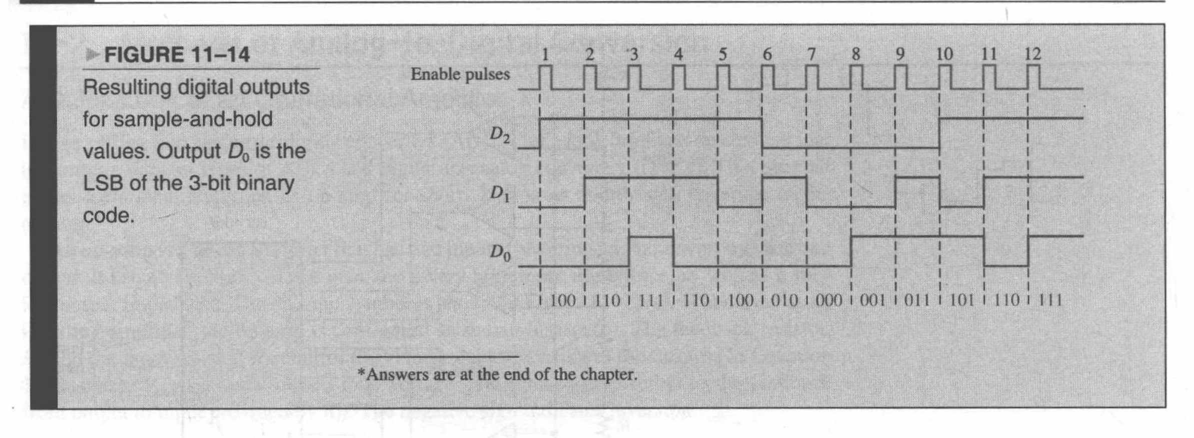


FIGURE 11–14 Resulting digital outputs for sample-and-hold values. Output D_0 is the LSB of the 3-bit binary code.

*Answers are at the end of the chapter.

Dual-Slope Analog-to-Digital Converter

A dual-slope ADC is common in digital voltmeters and other types of measurement instruments. A ramp generator (integrator) is used to produce the dual-slope characteristic. A block diagram of a dual-slope ADC is shown in Figure 11–15.

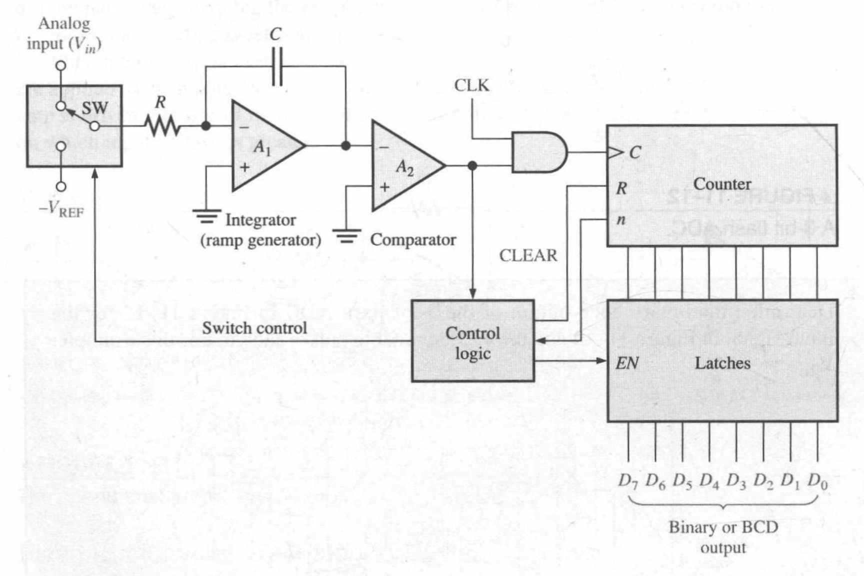


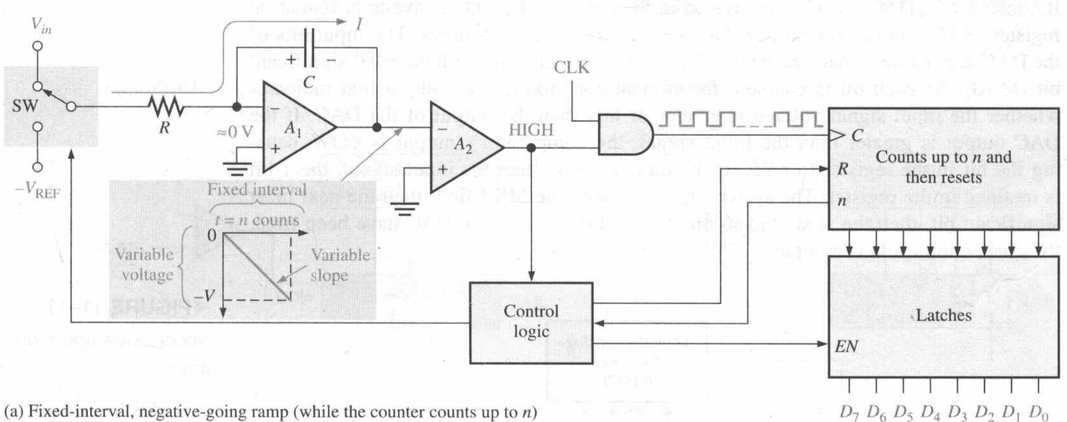
FIGURE 11–15 Basic dual-slope ADC.

Figure 11–16 illustrates dual-slope conversion. Start by assuming that the counter is reset and the output of the integrator is zero. Now assume that a positive input voltage is applied to the input through the switch (SW) as selected by the control logic. Since the inverting input of A_1 is at virtual ground, and assuming that V_{in} is constant for a period of time, there will be constant current through the input resistor R and therefore through the capacitor C. Capacitor C will charge linearly because the current is constant, and as a result, there will be a negative-going linear voltage ramp on the output of A_1, as illustrated in Figure 11–16(a).

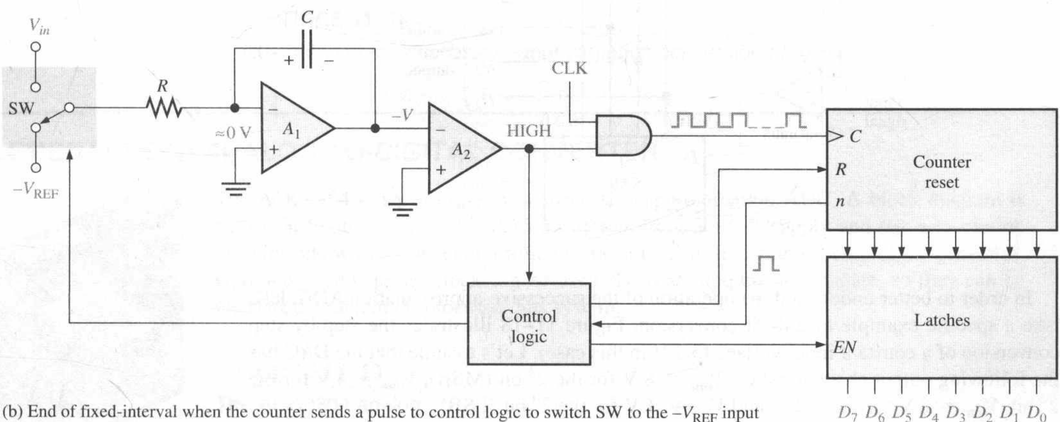
When the counter reaches a specified count (n), it will be reset (R), and the control logic will switch the negative reference voltage ($-V_{REF}$) to the input of A_1, as shown in Figure 11–16(b). At this point the capacitor is charged to a negative voltage ($-V$) proportional to the input analog voltage.

Now the capacitor discharges linearly because of the constant current from the $-V_{REF}$, as shown in Figure 11–16(c). This linear discharge produces a positive-going ramp on the A_1 output, starting at $-V$ and having a constant slope that is independent of the charge voltage. As the capacitor discharges, the counter advances from its RESET state. The time it

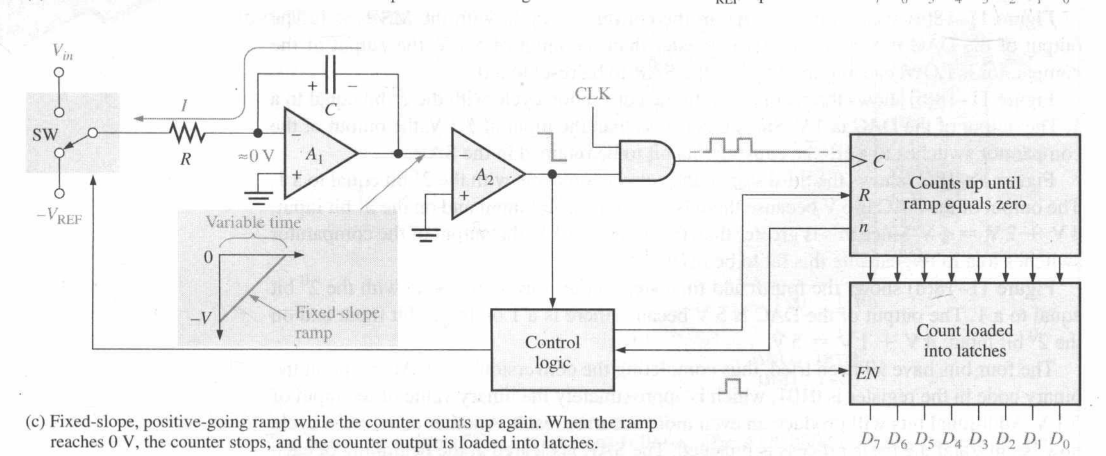
takes the capacitor to discharge to zero depends on the initial voltage $-V$ (proportional to V_{in}) because the discharge rate (slope) is constant. When the integrator (A_1) output voltage reaches zero, the comparator (A_2) switches to the LOW state and disables the clock to the counter. The binary count is latched, thus completing one conversion cycle. The binary count is proportional to V_{in} because the time it takes the capacitor to discharge depends only on $-V$, and the counter records this interval of time.



(a) Fixed-interval, negative-going ramp (while the counter counts up to n)



(b) End of fixed-interval when the counter sends a pulse to control logic to switch SW to the $-V_{REF}$ input



(c) Fixed-slope, positive-going ramp while the counter counts up again. When the ramp reaches 0 V, the counter stops, and the counter output is loaded into latches.

▲ **FIGURE 11–16**

Illustration of dual-slope conversion.

Successive-Approximation Analog-to-Digital Converter

One of the most widely used methods of analog-to-digital conversion is successive-approximation. It has a much faster conversion time than the dual-slope conversion, but it is slower than the flash method. It also has a fixed conversion time that is the same for any value of the analog input.

Figure 11–17 shows a basic block diagram of a 4-bit successive approximation ADC. It consists of a DAC (DACs are covered in Section 11–3), a successive-approximation register (SAR), and a comparator. The basic operation is as follows: The input bits of the DAC are enabled (made equal to a 1) one at a time, starting with the most significant bit (MSB). As each bit is enabled, the comparator produces an output that indicates whether the input signal voltage is greater or less than the output of the DAC. If the DAC output is greater than the input signal, the comparator's output is LOW, causing the bit in the register to reset. If the output is less than the input signal, the 1 bit is retained in the register. The system does this with the MSB first, then the next most significant bit, then the next, and so on. After all the bits of the DAC have been tried, the conversion cycle is complete.

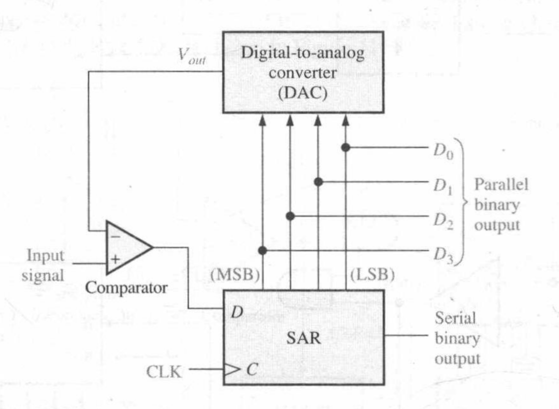


FIGURE 11–17 Successive-approximation ADC.

In order to better understand the operation of the successive-approximation ADC, let's take a specific example of a 4-bit conversion. Figure 11–18 illustrates the step-by-step conversion of a constant input voltage (5.1 V in this case). Let's assume that the DAC has the following output characteristics: V_{out} = 8 V for the 2^3 bit (MSB), V_{out} = 4 V for the 2^2 bit, V_{out} = 2 V for the 2^1 bit, and V_{out} = 1 V for the 2^0 bit (LSB).

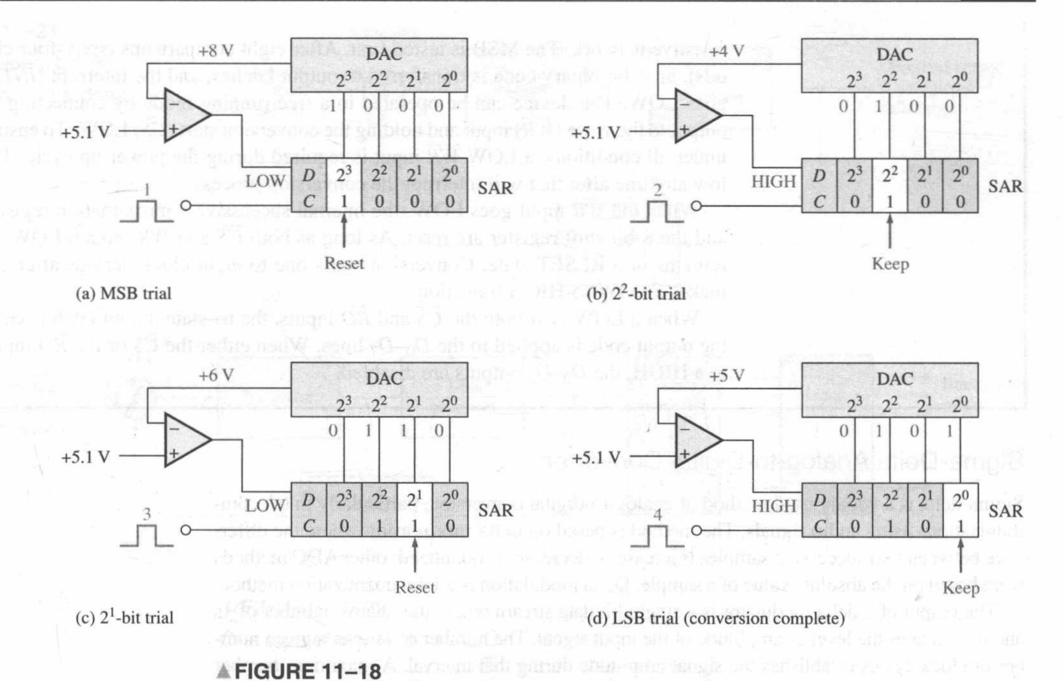
Figure 11–18(a) shows the first step in the conversion cycle with the MSB = 1. The output of the DAC is 8 V. Since this is greater than the input of 5.1 V, the output of the comparator is LOW, causing the MSB in the SAR to be reset to a 0.

Figure 11–18(b) shows the second step in the conversion cycle with the 2^2 bit equal to a 1. The output of the DAC is 4 V. Since this is less than the input of 5.1 V, the output of the comparator switches to a HIGH, causing this bit to be retained in the SAR.

Figure 11–18(c) shows the third step in the conversion cycle with the 2^1 bit equal to a 1. The output of the DAC is 6 V because there is a 1 on the 2^2 bit input and on the 2^1 bit input; 4 V + 2 V = 6 V. Since this is greater than the input of 5.1 V, the output of the comparator switches to a LOW, causing this bit to be reset to a 0.

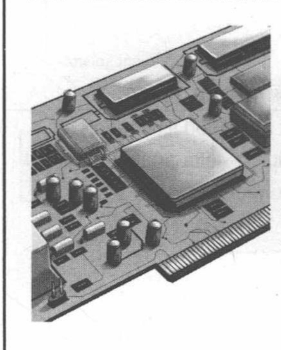
Figure 11–18(d) shows the fourth and final step in the conversion cycle with the 2^0 bit equal to a 1. The output of the DAC is 5 V because there is a 1 on the 2^2 bit input and on the 2^0 bit input; 4 V + 1 V = 5 V.

The four bits have all been tried, thus completing the conversion cycle. At this point the binary code in the register is 0101, which is approximately the binary value of the input of 5.1 V. Additional bits will produce an even more accurate result. Another conversion cycle now begins, and the basic process is repeated. The SAR is cleared at the beginning of each cycle.

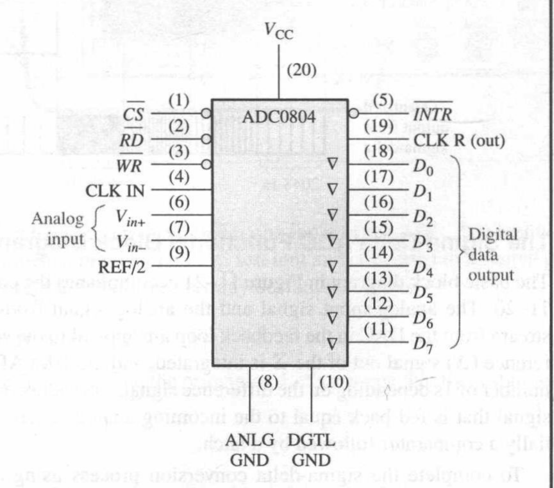


▲FIGURE 11–18
Illustration of the successive-approximation conversion process.

IMPLEMENTATION: ANALOG-TO-DIGITAL CONVERTER



The ADC0804 is an example of a successive-approximation ADC. A block diagram is shown in Figure 11–19. This device operates from a +5 V supply and has a resolution of eight bits with a conversion time of 100 μs. Also, it has an on-chip clock generator. Optionally, an external clock can be used. The data outputs are tri-state, so they can be interfaced with a microprocessor bus system.



►FIGURE 11–19
The ADC0804 analog-to-digital converter.

The basic operation of the device is as follows: The ADC0804 contains the equivalent of a 256-resistor DAC network. The successive-approximation logic sequences the network to match the analog differential input voltage ($V_{in+} - V_{in-}$) with an output from the

resistive network. The MSB is tested first. After eight comparisons (sixty-four clock periods), an 8-bit binary code is transferred to output latches, and the interrupt ($\overline{INTR}$) output goes LOW. The device can be operated in a free-running mode by connecting the $\overline{INTR}$ output to the write ($\overline{WR}$) input and holding the conversion start ($\overline{CS}$) LOW. To ensure startup under all conditions, a LOW $\overline{WR}$ input is required during the power-up cycle. Taking $\overline{CS}$ low anytime after that will interrupt the conversion process.

When the $\overline{WR}$ input goes LOW, the internal successive-approximation register (SAR) and the 8-bit shift register are reset. As long as both $\overline{CS}$ and $\overline{WR}$ remain LOW, the ADC remains in a RESET state. Conversion starts one to eight clock periods after $\overline{CS}$ or $\overline{WR}$ makes a LOW-to-HIGH transition.

When a LOW is at both the $\overline{CS}$ and $\overline{RD}$ inputs, the tri-state output latch is enabled and the output code is applied to the D_0–D_7 lines. When either the $\overline{CS}$ or the $\overline{RD}$ input returns to a HIGH, the D_0–D_7 outputs are disabled.

Sigma-Delta Analog-to-Digital Converter

Sigma-delta is a widely used method of analog-to-digital conversion, particularly in telecommunications using audio signals. The method is based on **delta modulation** where the difference between two successive samples (increase or decrease) is quantized; other ADC methods were based on the absolute value of a sample. Delta modulation is a 1-bit quantization method.

The output of a delta modulator is a single-bit data stream where the relative number of 1s and 0s indicates the level or amplitude of the input signal. The number of 1s over a given number of clock cycles establishes the signal amplitude during that interval. A maximum number of 1s corresponds to the maximum positive input voltage. A number of 1s equal to one-half the maximum corresponds to an input voltage of zero. No 1s (all 0s) corresponds to the maximum negative input voltage. This is illustrated in a simplified way in Figure 11–20. For example, assume that 4096 1s occur during the interval when the input signal is a positive maximum. Since zero is the midpoint of the dynamic range of the input signal, 2048 1s occur during the interval when the input signal is zero. There are no 1s during the interval when the input signal is a negative maximum. For signal levels in between, the number of 1s is proportional to the level.

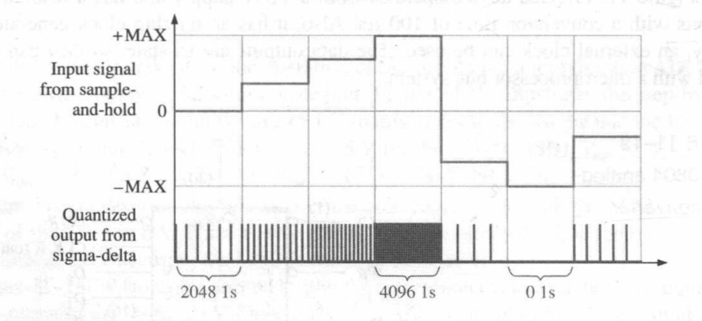
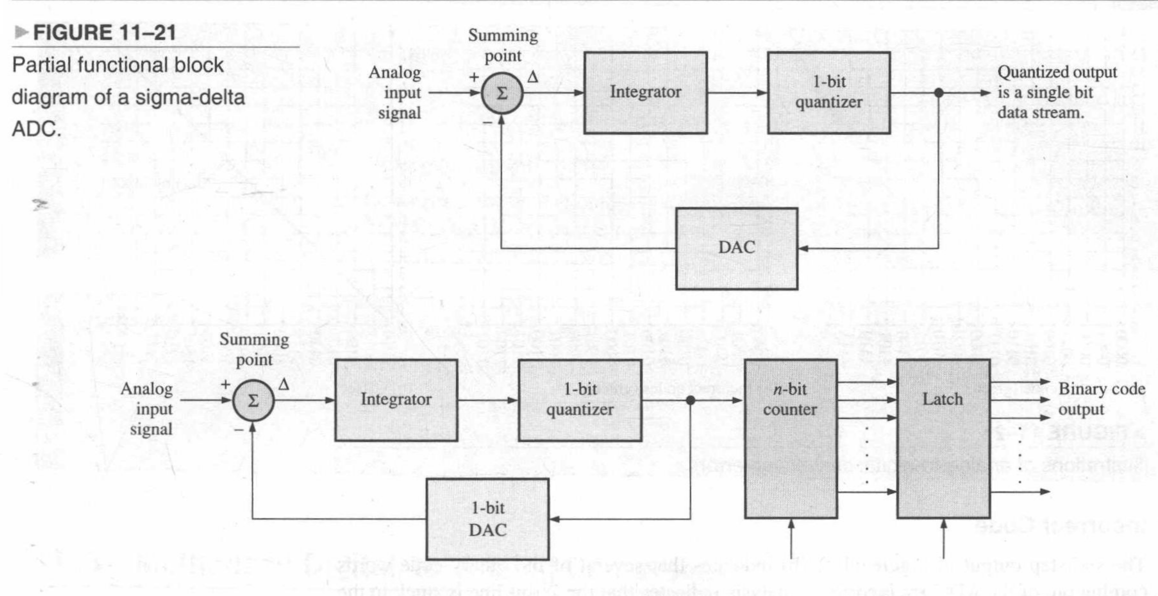


FIGURE 11–20 A simplified illustration of sigma-delta analog-to-digital conversion.

The Sigma-Delta ADC Functional Block Diagram

The basic block diagram in Figure 11–21 accomplishes the conversion illustrated in Figure 11–20. The analog input signal and the analog signal from the converted quantized bit stream from the DAC in the feedback loop are applied to the summation (Σ) point. The difference (Δ) signal out of the Σ is integrated, and the 1-bit ADC increases or decreases the number of 1s depending on the difference signal. This action attempts to keep the quantized signal that is fed back equal to the incoming analog signal. The 1-bit quantizer is essentially a comparator followed by a latch.

To complete the sigma-delta conversion process using one particular approach, the single bit data stream is converted to a series of binary codes, as shown in Figure 11–22. The counter counts the 1s in the quantized data stream for successive intervals. The code in the counter then represents the amplitude of the analog input signal for each interval. These codes are shifted out into the latch for temporary storage. What comes out of the latch is a series of n-bit codes, which completely represent the analog signal.



▶ **FIGURE 11–21**

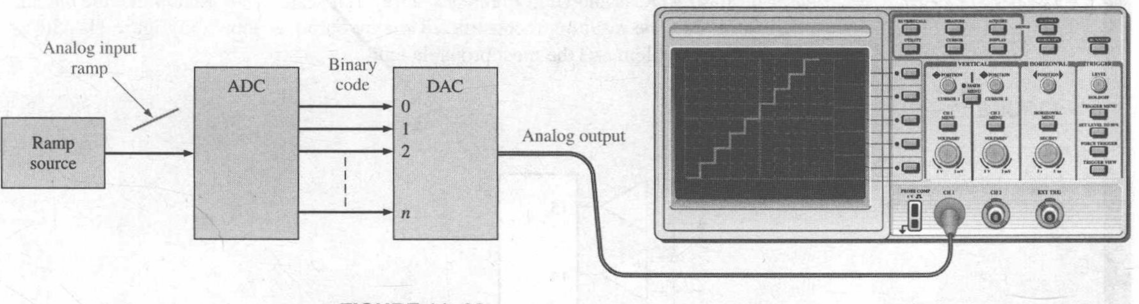
Partial functional block diagram of a sigma-delta ADC.

▲ **FIGURE 11–22**

One type of sigma-delta ADC.

Testing Analog-to-Digital Converters

One method for testing ADCs is shown in Figure 11–23. A DAC is used as part of the test setup to convert the ADC output back to analog form for comparison with the test input.



▲ **FIGURE 11–23**

A method for testing ADCs.

A test input in the form of a linear ramp is applied to the input of the ADC. The resulting binary output sequence is then applied to the DAC test unit and converted to a stairstep ramp. The input and output ramps are compared for any deviation.

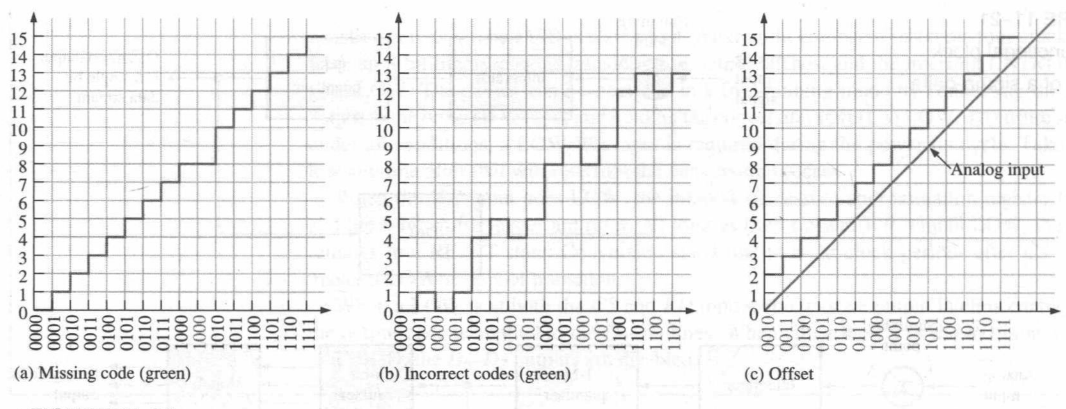
Analog-to-Digital Conversion Errors

Again, a 4-bit conversion is used to illustrate the principles. Let's assume that the test input is an ideal linear ramp.

Missing Code

The stairstep output in Figure 11–24(a) indicates that the binary code 1001 does not appear on the output of the ADC. Notice that the 1000 value stays for two intervals and then the output jumps to the 1010 value.

In a flash ADC, for example, a failure of one of the op-amp comparators can cause a missing-code error.



(a) Missing code (green) (b) Incorrect codes (green) (c) Offset

▲ **FIGURE 11–24**

Illustrations of analog-to-digital conversion errors.

Incorrect Code

The stairstep output in Figure 11–24(b) indicates that several of the binary code words coming out of the ADC are incorrect. Analysis indicates that the 2^1-bit line is stuck in the LOW (0) state in this particular case.

Offset

Offset conditions are shown in 11–24(c). In this situation the ADC interprets the analog input voltage as greater than its actual value.

EXAMPLE 11-2

A 4-bit flash ADC is shown in Figure 11–25(a). It is tested with a setup like the one in Figure 11–23. The resulting reconstructed analog output is shown in Figure 11–25(b). Identify the problem and the most probable fault.

► **FIGURE 11–25**

V_{REF}

Analog ramp input

15 14 1 2 4 8 3 2 1 0 *EN*

(a) (b)

Solution

The binary code 0011 is missing from the ADC output, as indicated by the missing step. Most likely, the output of comparator 3 is stuck in its inactive state (LOW).

Related Problem

Reconstruct the analog output in a test setup like in Figure 11–23 if the ADC in Figure 11–25(a) has comparator 8 stuck in the HIGH output state.

SECTION 11-2 CHECKUP

1. What is the fastest method of analog-to-digital conversion?
2. Which analog-to-digital conversion method produces a single-bit data stream?
3. Does the successive-approximation converter have a fixed conversion time?
4. Name two types of output errors in an ADC.

11-3 Methods of Digital-to-Analog Conversion

Binary-Weighted-Input Digital-to-Analog Converter

One method of digital-to-analog conversion uses a resistor network with resistance values that represent the binary weights of the input bits of the digital code. Figure 11–26 shows a 4-bit DAC of this type. Each of the input resistors will either have current or have no current, depending on the input voltage level. If the input voltage is zero (binary 0), the current is also zero. If the input voltage is HIGH (binary 1), the amount of current depends on the input resistor value and is different for each input resistor, as indicated in the figure.

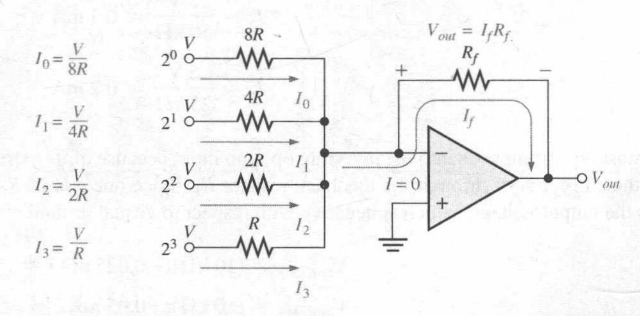


FIGURE 11–26
A 4-bit DAC with binary-weighted inputs.

Since there is practically no current into the op-amp inverting (−) input, all of the input currents sum together and go through R_f. Since the inverting input is at 0 V (virtual ground), the drop across R_f is equal to the output voltage, so $V_{out} = I_f R_f$.

The values of the input resistors are chosen to be inversely proportional to the binary weights of the corresponding input bits. The lowest-value resistor (R) corresponds to the highest binary-weighted input (2^3). The other resistors are multiples of R (that is, $2R$, $4R$, and $8R$) and correspond to the binary weights 2^2, 2^1, and 2^0, respectively. The input currents are also proportional to the binary weights. Thus, the output voltage is proportional to the sum of the binary weights because the sum of the input currents is through R_f.

Disadvantages of this type of DAC are the number of different resistor values and the fact that the voltage levels must be exactly the same for all inputs. For example, an 8-bit converter requires eight resistors, ranging from some value of R to $128R$ in binary-weighted steps. This range of resistors requires tolerances of one part in 255 (less than 0.5%) to accurately convert the input, making this type of DAC very difficult to mass-produce.

EXAMPLE 11-3

Determine the output of the DAC in Figure 11–27(a) if the waveforms representing a sequence of 4-bit numbers in Figure 11–27(b) are applied to the inputs. Input D_0 is the least significant bit (LSB).

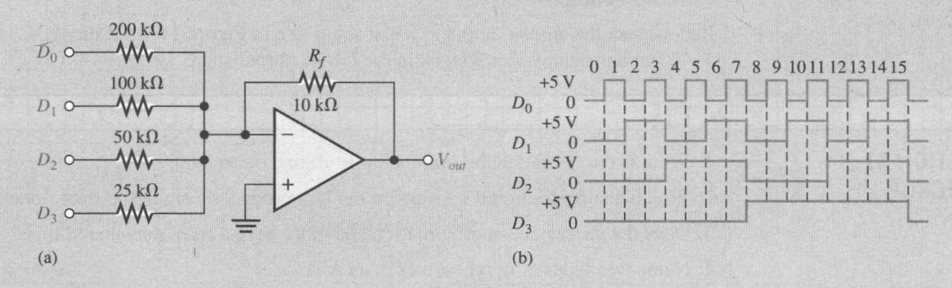


FIGURE 11–27

Solution

First, determine the current for each of the weighted inputs. Since the inverting (−) input of the op-amp is at 0 V (virtual ground) and a binary 1 corresponds to +5 V, the current through any of the input resistors is 5 V divided by the resistance value.

$$I_0 = \frac{5\text{ V}}{200\text{ k}\Omega} = 0.025\text{ mA}$$

$$I_1 = \frac{5\text{ V}}{100\text{ k}\Omega} = 0.05\text{ mA}$$

$$I_2 = \frac{5\text{ V}}{50\text{ k}\Omega} = 0.1\text{ mA}$$

$$I_3 = \frac{5\text{ V}}{25\text{ k}\Omega} = 0.2\text{ mA}$$

Almost no current goes into the inverting op-amp input because of its extremely high impedance. Therefore, assume that all of the current goes through the feedback resistor R_f. Since one end of R_f is at 0 V (virtual ground), the drop across R_f equals the output voltage, which is negative with respect to virtual ground.

$$V_{out(D0)} = (10\text{ k}\Omega)(-0.025\text{ mA}) = -0.25\text{ V}$$

$$V_{out(D1)} = (10\text{ k}\Omega)(-0.05\text{ mA}) = -0.5\text{ V}$$

$$V_{out(D2)} = (10\text{ k}\Omega)(-0.1\text{ mA}) = -1\text{ V}$$

$$V_{out(D3)} = (10\text{ k}\Omega)(-0.2\text{ mA}) = -2\text{ V}$$

From Figure 11–27(b), the first binary input code is 0000, which produces an output voltage of 0 V. The next input code is 0001, which produces an output voltage of −0.25 V. The next code is 0010, which produces an output voltage of −0.5 V. The next code is 0011, which produces an output voltage of −0.25 V + −0.5 V = −0.75 V. Each successive binary code increases the output voltage by −0.25 V, so for this particular straight binary sequence on the inputs, the output is a stairstep waveform going from 0 V to −3.75 V in −0.25 V steps. This is shown in Figure 11–28.

Related Problem

Reverse the input waveforms to the DAC in Figure 11–27 (D_3 to D_0, D_2 to D_1, D_1 to D_2, D_0 to D_3) and determine the output.

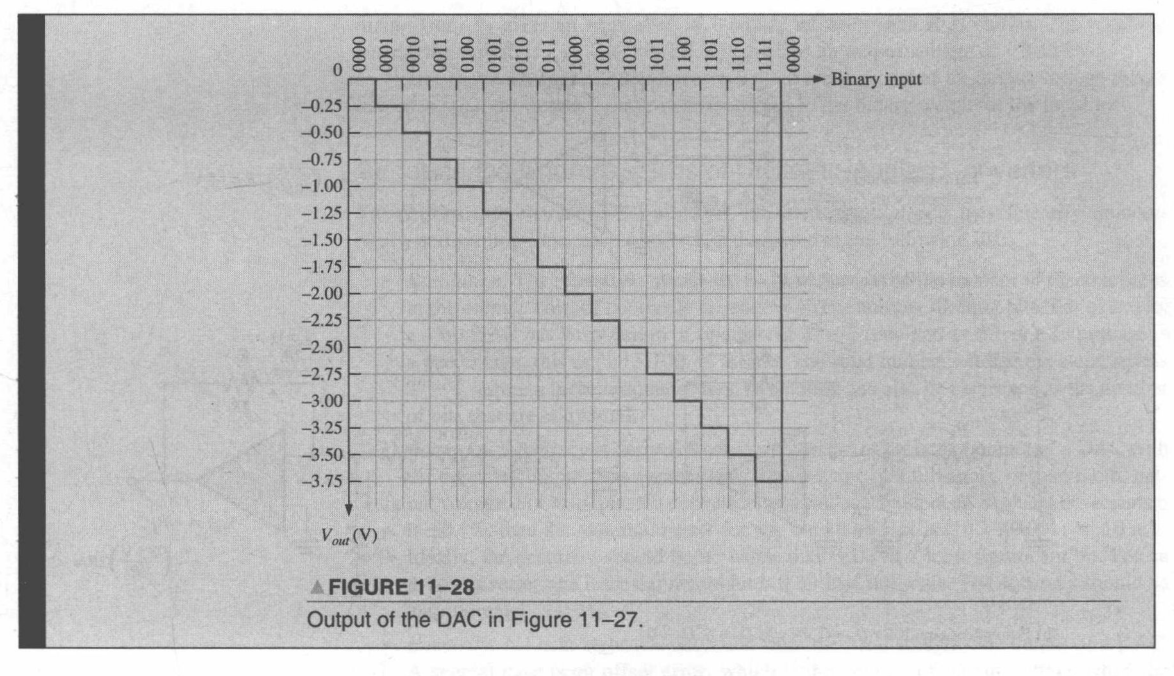
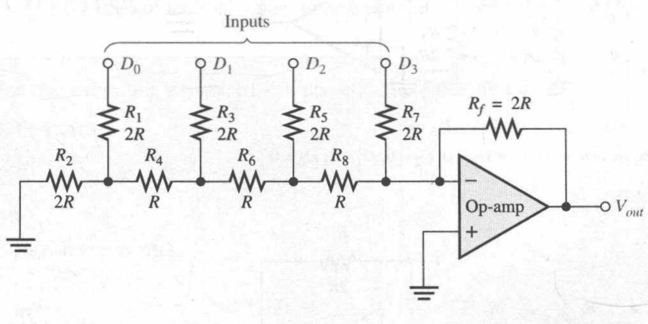


FIGURE 11–28
Output of the DAC in Figure 11–27.

The *R*/2*R* Ladder Digital-to-Analog Converter

Another method of digital-to-analog conversion is the *R*/2*R* ladder, as shown in Figure 11–29 for four bits. It overcomes one of the problems in the binary-weighted-input DAC in that it requires only two resistor values.

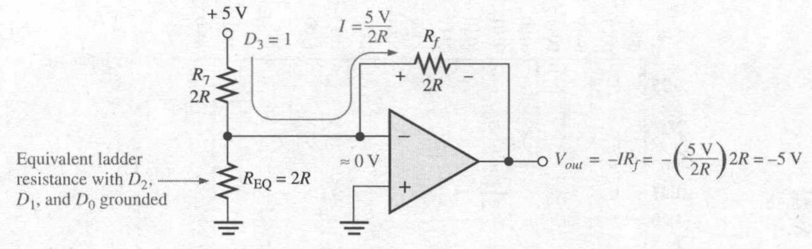
FIGURE 11–29
An *R*/2*R* ladder DAC.



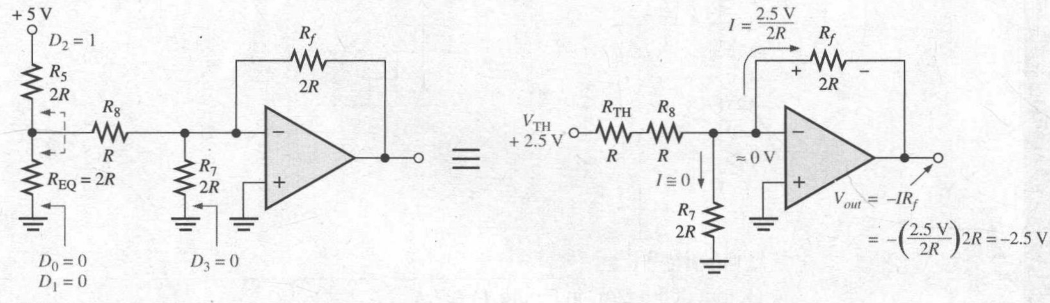
Start by assuming that the D_3 input is HIGH (+5 V) and the others are LOW (ground, 0 V). This condition represents the binary number 1000. A circuit analysis will show that this reduces to the equivalent form shown in Figure 11–30(a). Essentially no current goes through the 2*R* equivalent resistance because the inverting input is at virtual ground. Thus, all of the current ($I = 5\text{ V}/2R$) through R_7 also goes through R_f, and the output voltage is -5 V. The operational amplifier keeps the inverting ($-$) input near zero volts (≈ 0 V) because of negative feedback. Therefore, all current goes through R_f rather than into the inverting input.

Figure 11–30(b) shows the equivalent circuit when the D_2 input is at +5 V and the others are at ground. This condition represents 0100. If we thevenize* looking from R_8, we get 2.5 V in series with *R*, as shown. This results in a current through R_f of $I = 2.5\text{ V}/2R$, which gives an output voltage of -2.5 V. Keep in mind that there is no current into the op-amp inverting input and that there is no current through the equivalent resistance to ground because it has 0 V across it, due to the virtual ground.

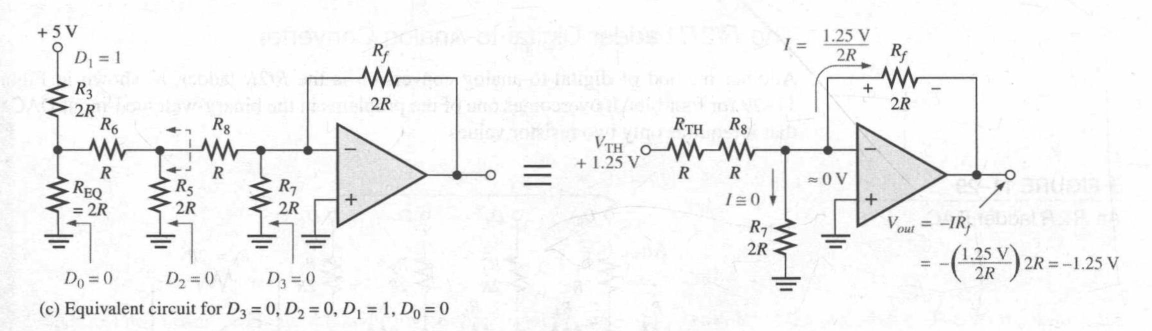
* Thevenin's theorem states that any circuit can be reduced to an equivalent voltage source in series with an equivalent resistance.



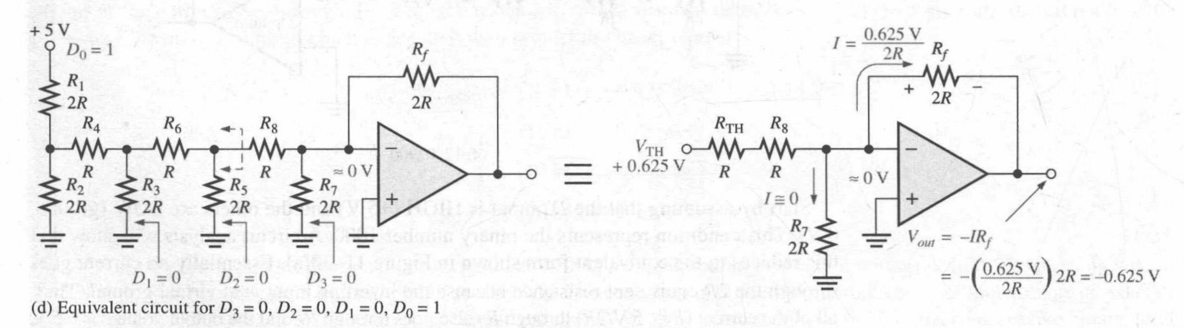
(a) Equivalent circuit for $D_3 = 1, D_2 = 0, D_1 = 0, D_0 = 0$



(b) Equivalent circuit for $D_3 = 0, D_2 = 1, D_1 = 0, D_0 = 0$



(c) Equivalent circuit for $D_3 = 0, D_2 = 0, D_1 = 1, D_0 = 0$



(d) Equivalent circuit for $D_3 = 0, D_2 = 0, D_1 = 0, D_0 = 1$

▲ **FIGURE 11–30**

Analysis of the *R*/2*R* ladder DAC.

Figure 11–30(c) shows the equivalent circuit when the D_1 input is at +5 V and the others are at ground. This condition represents 0010. Again thevenizing looking from R_8, you get 1.25 V in series with R as shown. This results in a current through R_f of $I = 1.25\text{ V}/2R$, which gives an output voltage of -1.25 V.

In part (d) of Figure 11–30, the equivalent circuit representing the case where D_0 is at +5 V and the other inputs are at ground is shown. This condition represents 0001. Theve-

nizing from R_8 gives an equivalent of 0.625 V in series with R as shown. The resulting current through R_f is $I = 0.625\text{ V}/2R$, which gives an output voltage of -0.625 V.

Notice that each successively lower-weighted input produces an output voltage that is halved, so that the output voltage is proportional to the binary weight of the input bits.

Performance Characteristics of Digital-to-Analog Converters

The performance characteristics of a DAC include resolution, accuracy, linearity, monotonicity, and settling time, each of which is discussed in the following list:

- *Resolution.* The resolution of a DAC is the reciprocal of the number of discrete steps in the output. This, of course, is dependent on the number of input bits. For example, a 4-bit DAC has a resolution of one part in $2^4 - 1$ (one part in fifteen). Expressed as a percentage, this is $(1/15)100 = 6.67\%$. The total number of discrete steps equals $2^n - 1$, where n is the number of bits. Resolution can also be expressed as the number of bits that are converted.
- *Accuracy.* Accuracy is derived from a comparison of the actual output of a DAC with the expected output. It is expressed as a percentage of a full-scale, or maximum, output voltage. For example, if a converter has a full-scale output of 10 V and the accuracy is $\pm 0.1\%$, then the maximum error for any output voltage is $(10\text{ V})(0.001) = 10$ mV. Ideally, the accuracy should be no worse than $\pm 1/2$ of a least significant bit. For an 8-bit converter, the least significant bit is 0.39% of full scale. The accuracy should be approximately $\pm 0.2\%$.
- *Linearity.* A linear error is a deviation from the ideal straight-line output of a DAC. A special case is an offset error, which is the amount of output voltage when the input bits are all zeros.
- *Monotonicity.* A DAC is **monotonic** if it does not take any reverse steps when it is sequenced over its entire range of input bits.
- *Settling time.* Settling time is normally defined as the time it takes a DAC to settle within $\pm 1/2$ LSB of its final value when a change occurs in the input code.

EXAMPLE 11-4

Determine the resolution, expressed as a percentage, of the following:

(a) an 8-bit DAC

(b) a 12-bit DAC

Solution

(a) For the 8-bit converter,

$$\frac{1}{2^8 - 1} \times 100 = \frac{1}{255} \times 100 = \mathbf{0.392\%}$$

(b) For the 12-bit converter,

$$\frac{1}{2^{12} - 1} \times 100 = \frac{1}{4095} \times 100 = \mathbf{0.0244\%}$$

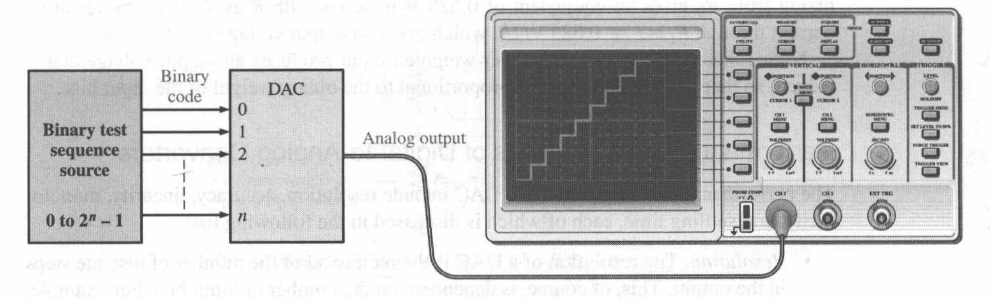
Related Problem

Calculate the resolution for a 16-bit DAC.

*Thevenin's theorem states that any circuit can be reduced to an equivalent voltage source in series with an equivalent resistance.

Testing Digital-to-Analog Converters

The concept of DAC testing is illustrated in Figure 11–31. In this basic method, a sequence of binary codes is applied to the inputs, and the resulting output is observed. The binary code sequence extends over the full range of values from 0 to $2^n - 1$ in ascending order, where n is the number of bits.

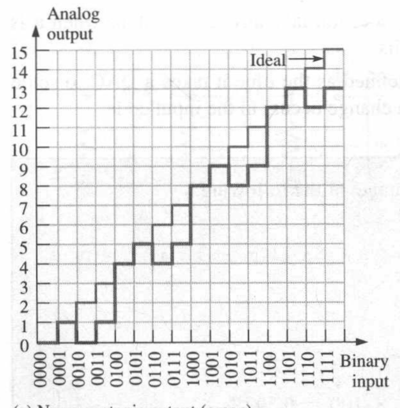


▲FIGURE 11–31
Basic test setup for a DAC.

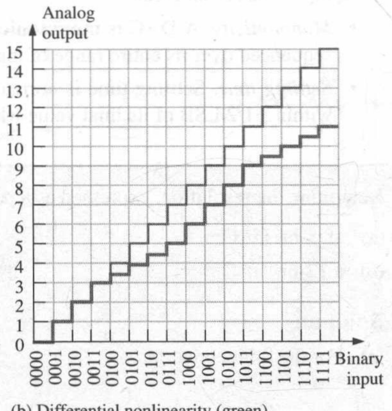
The ideal output is a straight-line stairstep as indicated. As the number of bits in the binary code is increased, the resolution is improved. That is, the number of discrete steps increases, and the output approaches a straight-line linear ramp.

Digital-to-Analog Conversion Errors

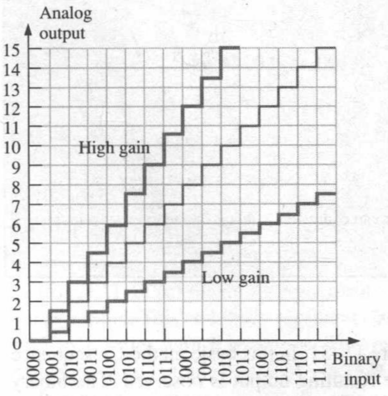
Several digital-to-analog conversion errors to be checked for are shown in Figure 11–32, which uses a 4-bit conversion for illustration purposes. A 4-bit conversion produces fifteen discrete steps. Each graph in the figure includes an ideal stairstep ramp for comparison with the faulty outputs.



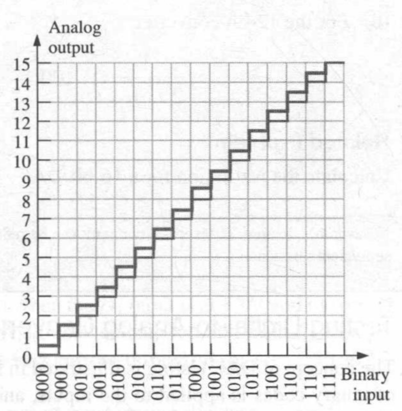
(a) Nonmonotonic output (green)



(b) Differential nonlinearity (green)



(c) High and low gains (green)



(d) Offset error (green)

◀FIGURE 11–32
Illustrations of several digital-to-analog conversion errors.

Nonmonotonicity

The step reversals in Figure 11–32(a) indicate nonmonotonic performance, which is a form of nonlinearity. In this particular case, the error occurs because the 2^1 bit in the binary code is interpreted as a constant 0. That is, a short is causing the bit input line to be stuck LOW.

Differential Nonlinearity

Figure 11–32(b) illustrates differential nonlinearity in which the step amplitude is less than it should be for certain input codes. This particular output could be caused by the 2^2 bit having an insufficient weight, perhaps because of a faulty input resistor. We could also see steps with amplitudes greater than normal if a particular binary weight were greater than it should be.

Low or High Gain

Output errors caused by low or high gain are illustrated in Figure 11–32(c). In the case of low gain, all of the step amplitudes are less than ideal. In the case of high gain, all of the step amplitudes are greater than ideal. This situation may be caused by a faulty feedback resistor in the op-amp circuit.

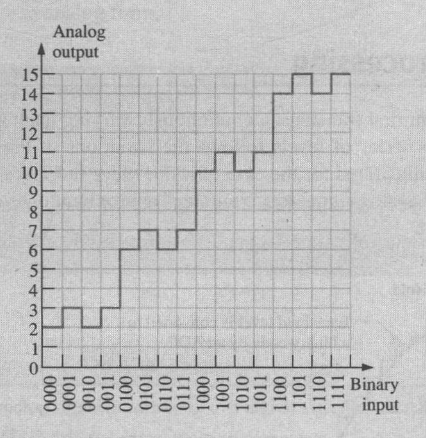
Offset Error

An offset error is illustrated in Figure 11–32(d). Notice that when the binary input is 0000, the output voltage is nonzero and that this amount of offset is the same for all steps in the conversion. A faulty op-amp may be the culprit in this situation.

EXAMPLE 11-5

The DAC output in Figure 11–33 is observed when a straight 4-bit binary sequence is applied to the inputs. Identify the type of error, and suggest an approach to isolate the fault.

▶ FIGURE 11–33



Solution

The DAC in this case is nonmonotonic. Analysis of the output reveals that the device is converting the following sequence, rather than the actual binary sequence applied to the inputs.

0010, 0011, 0010, 0011, 0110, 0111, 0110, 0111, 1010, 1011, 1010, 1011, 1110, 1111,1110, 1111

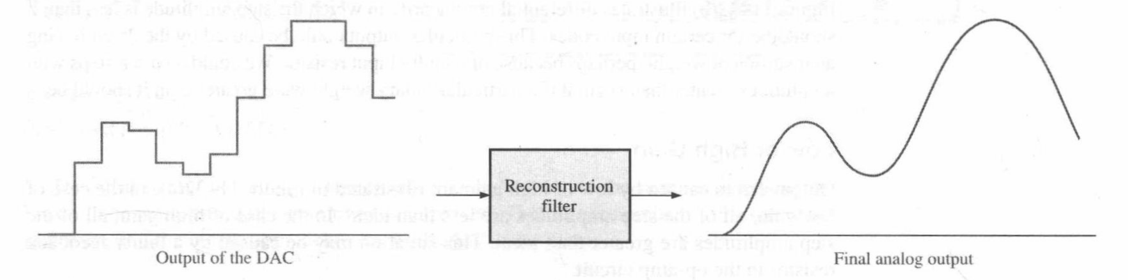
Apparently, the 2^1 bit is stuck in the HIGH (1) state. To find the problem, first monitor the bit input pin to the device. If it is changing states, the fault is internal to the DAC and it should be replaced. If the external pin is not changing states and is always HIGH, check for an external short to $+V$ that may be caused by a solder bridge somewhere on the circuit board.

Related Problem

Determine the output of a DAC when a straight 4-bit binary sequence is applied to the inputs and the 2^0 bit is stuck HIGH.

The Reconstruction Filter

The output of the DAC is a "stairstep" approximation of the original analog signal after it has been processed by the **digital signal processor (DSP)**, which is a special type of microprosessor that processes data in real time. The purpose of the low-pass reconstruction filter (sometimes called a postfilter) is to smooth out the DAC output by eliminating the higher frequency content that results from the fast transitions of the "stairsteps," as roughly illustrated in Figure 11–34.



▲FIGURE 11–34

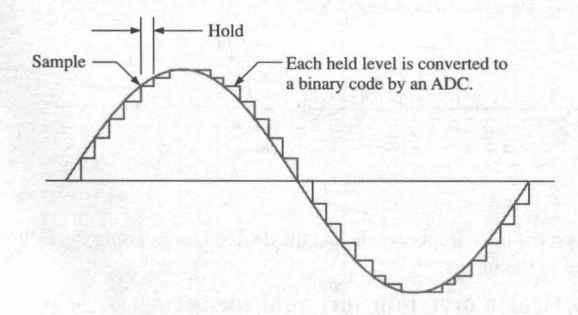
The reconstruction filter smooths the output of the DAC.

SECTION 11-3 CHECKUP

1. What is the disadvantage of the DAC with binary-weighted inputs?
2. What is the resolution of a 4-bit DAC?
3. How do you detect nonmonotonic behavior in a DAC?
4. What effect does low gain have on a DAC output?

11-4 Digital Signal Processing

A digital signal processing system first translates a continuously varying analog signal into a series of discrete levels. This series of levels follows the variations of the analog signal and resembles a staircase, as illustrated for the case of a sine wave in Figure 11–35. The process of changing the original analog signal to a "stairstep" approximation is accomplished by a sample-and-hold circuit.



◄FIGURE 11–35

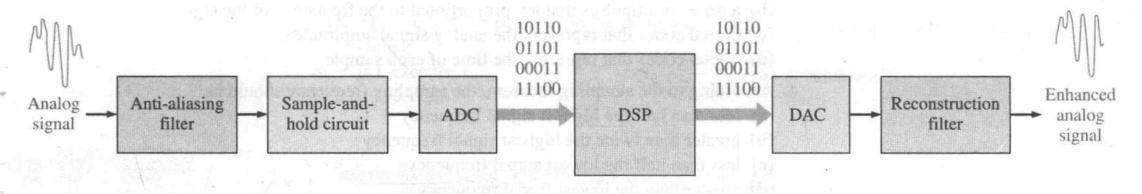
An original analog signal (sine wave) and its "stairstep" approximation.

Next, the "stairstep" approximation is quantized into binary codes that represent each discrete step on the "stairsteps" by a process called analog-to-digital (A/D) conversion. The circuit that performs A/D conversion is an analog-to-digital converter (ADC).

Once the analog signal has been converted to a binary coded form, it is applied to a DSP (digital signal processor). The DSP can perform various operations on the incoming data, such as removing unwanted interference, increasing the amplitude of some signal

frequencies and reducing others, encoding the data for secure transmissions, and detecting and correcting errors in transmitted codes. DSPs make possible, among many other things, the cleanup of sound recordings, the removal of echos from communications lines, the enhancement of images from CT scans for better medical diagnosis, and the scrambling of cellular phone conversations for privacy.

After a DSP processes a signal, the signal can be converted back to an enhanced version of the original analog signal. This is accomplished by a digital-to-analog converter (DAC). Figure 11–36 shows a basic block diagram of a typical digital signal processing system.



▲**FIGURE 11–36**

Basic block diagram of a typical digital signal processing system.

DSPs are actually a specialized type of microprocessor but are different from general-purpose microprocessors in a couple of significant ways. Typically, microprocessors are designed for general-purpose functions and operate with large software packages. DSPs are used for special-purpose applications; they are very fast number crunchers that must work in real time by processing information as it happens using specialized algorithms (programs). The analog-to-digital converter (ADC) in a system must take samples of the incoming analog data often enough to catch all the relevant fluctuations in the signal amplitude, and the DSP must keep pace with the sampling rate of the ADC by doing its calculations as fast as the sampled data are received. Once the digital data are processed by the DSP, they go to the digital-to-analog converter (DAC) and reconstruction filter for conversion back to analog form.

SECTION 11-4 CHECKUP

1. What does DSP stand for?
2. What does ADC stand for?
3. What does DAC stand for?
4. An analog signal is changed to a binary coded form by what circuit?
5. A binary coded signal is changed to analog form by what circuit?

TRUE/FALSE QUIZ

Answers are at the end of the chapter.

1. An analog signal is converted to a digital signal by an ADC.
2. A DAC is a digital approximation computer.
3. The Nyquist frequency is twice the sampling frequency.
4. A higher sampling rate is more accurate than a lower sampling rate for a given analog signal.
5. Resolution is the number of bits used by an analog-to-digital converter.
6. Successful approximation is an analog-to-digital conversion method.
7. Delta modulation is based on the difference of two successive samples.
8. Two types of DAC are the binary-weighted input and the *R*/2*R* ladder.
9. The process of converting an analog value to a code is called *quantization*.
10. A flash ADC differs from a simultaneous ADC.

SELF-TEST

Answers are at the end of the chapter.

1. An ADC is an
   (a) alphanumeric data code
   (b) analog-to-digital converter
   (c) analog device carrier
   (d) analog-to-digital comparator
2. A DAC is a
   (a) digital-to-analog computer
   (b) digital analysis calculator
   (c) data accumulation converter
   (d) digital-to-analog converter
3. Sampling of an analog signal produces
   (a) a series of impulses that are proportional to the amplitude of the signal
   (b) a series of impulses that are proportional to the frequency of the signal
   (c) digital codes that represent the analog signal amplitude
   (d) digital codes that represent the time of each sample
4. According to the sampling theorem, the sampling frequency should be
   (a) less than half the highest signal frequency
   (b) greater than twice the highest signal frequency
   (c) less than half the lowest signal frequency
   (d) greater than the lowest signal frequency
5. A hold action occurs
   (a) before each sample
   (b) during each sample
   (c) after the analog-to-digital conversion
   (d) immediately after a sample
6. The quantization process
   (a) converts the sample-and-hold output to binary code
   (b) converts a sample impulse to a level
   (c) converts a sequence of binary codes to a reconstructed analog signal
   (d) filters out unwanted frequencies before sampling takes place
7. Generally, an analog signal can be reconstructed more accurately with
   (a) more quantization levels
   (b) fewer quantization levels
   (c) a higher sampling frequency
   (d) a lower sampling frequency
   (e) either answer (a) or (c)
8. A flash ADC uses
   (a) counters
   (b) op-amps
   (c) an integrator
   (d) flip-flops
   (e) answers (a) and (c)
9. A dual-slope ADC uses
   (a) a counter
   (b) op-amps
   (c) an integrator
   (d) a differentiator
   (e) answers (a) and (c)
10. The output of a sigma-delta ADC is
    (a) parallel binary codes
    (b) multiple-bit data
    (c) single-bit data
    (d) a difference voltage
11. In a binary-weighted DAC, the resistors on the inputs
    (a) determine the amplitude of the analog signal
    (b) determine the weights of the digital inputs
    (c) limit the power consumption
    (d) prevent loading on the source
12. In an $R/2R$ DAC, there are
    (a) four values of resistors
    (b) one resistor value
    (c) two resistor values
    (d) a number of resistor values equal to the number of inputs
13. A digital signal processing system usually operates in
    (a) real time
    (b) imaginary time
    (c) compressed time
    (d) computer time
14. The term *Harvard architecture* means
    (a) a CPU and a main memory
    (b) a CPU and two data memories
    (c) a CPU, a program memory, and a data memory
    (d) a CPU and two register files

15. The minimum number of general-purpose registers in the TMS320C6000 series DSPs is
(a) 32 (b) 64
(c) 16 (d) 8

16. The two internal memories in the TMS320C6000 series each have a capacity of
(a) 1 MB (b) 512 kB
(c) 64 kB (d) 32 kB

17. In the TMS320C6000 series pipeline operation, the number of instructions processed simultaneously is
(a) eight (b) four
(c) two (d) one

18. The stage of the pipeline operation in which instructions are retrieved from the memory is called
(a) execute (b) accumulate
(c) decode (d) fetch

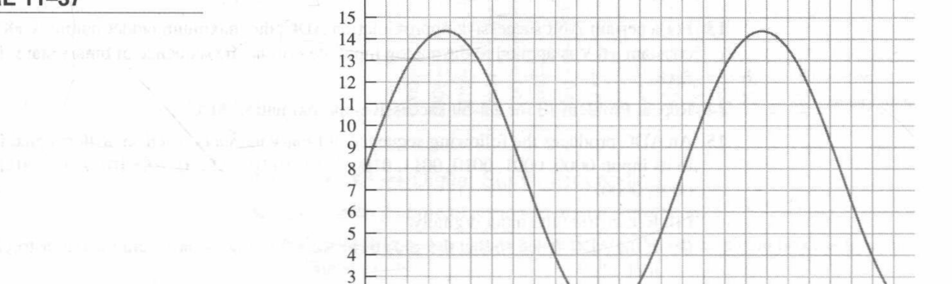
PROBLEMS

Answers to odd-numbered problems are at the end of the book.

Section 11–1 Analog-to-Digital Conversion

1. The waveform shown in Figure 11–37 is applied to a sampling circuit and is sampled every 3 ms. Show the output of the sampling circuit. Assume a one-to-one voltage correspondence between the input and output.

▶ FIGURE 11–37

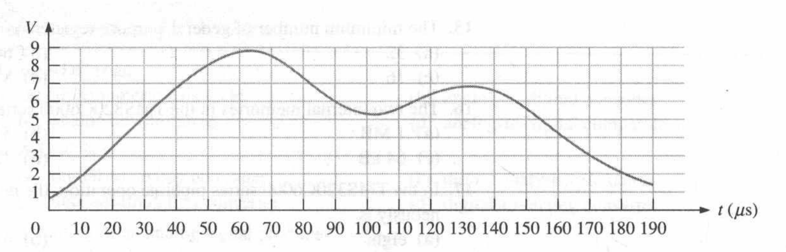


2. The output of the sampling circuit in Problem 1 is applied to a hold circuit. Show the output of the hold circuit.
3. If the output of the hold circuit in Problem 2 is quantized using two bits, what is the resulting sequence of binary codes?
4. Repeat Problem 3 using 4-bit quantization.
5. (a) Reconstruct the analog signal from the 2-bit quantization in Problem 3.
(b) Reconstruct the analog signal from the 4-bit quantization in Problem 4.
6. Graph the analog function represented by the following sequence of binary numbers: 1111, 1110, 1101, 1100, 1010, 1001, 1000, 0111, 0110, 0101, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1100, 1100, 1011, 1010, 1001.

Section 11–2 Methods of Analog-to-Digital Conversion

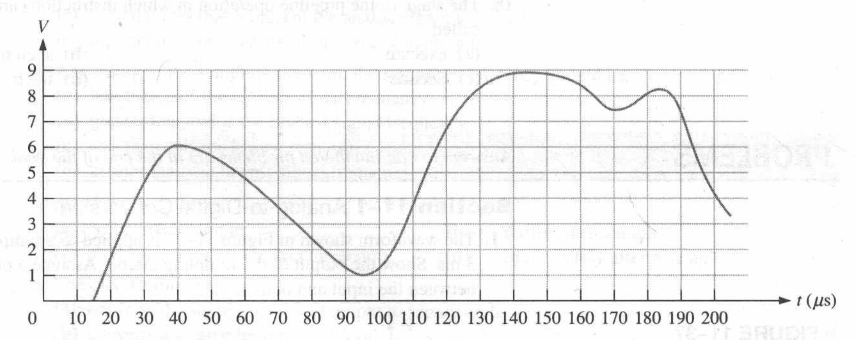
7. The input voltage to a certain op-amp inverting amplifier is 10 mV, and the output is 2 V. What is the closed-loop voltage gain?
8. To achieve a closed-loop voltage gain of 330 with an inverting amplifier, what value of feedback resistor do you use if $R_i = 1.0\ \text{k}\Omega$?
9. What is the gain of an inverting amplifier that uses a 47 kΩ feedback resistor if the input resistor is 2.2 kΩ?
10. How many comparators are required to form an 8-bit flash converter?
11. Determine the binary output code of a 3-bit flash ADC for the analog input signal in Figure 11–38.

▶ FIGURE 11–38



12. Repeat Problem 11 for the analog waveform in Figure 11–39.

▶ FIGURE 11–39



13. For a certain 2-bit successive-approximation ADC, the maximum ladder output is +8 V. If a constant +6 V is applied to the analog input, determine the sequence of binary states for the SAR.

14. Repeat Problem 13 for a 4-bit successive-approximation ADC.

15. An ADC produces the following sequence of binary numbers when an analog signal is applied to its input: 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 0110, 0101, 0100, 0011, 0010, 0001, 0000.

    (a) Reconstruct the input digitally.

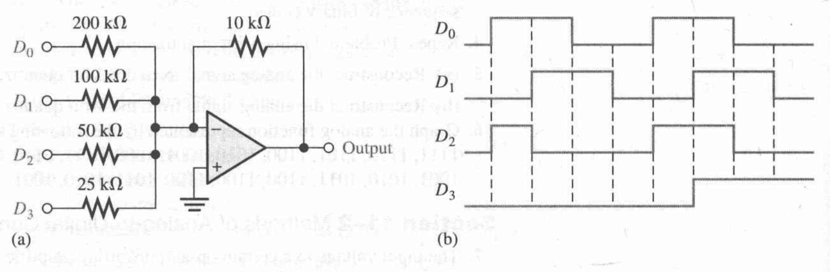
    (b) If the ADC failed so that the code 0111 were missing, what would the reconstructed output look like?

Section 11–3 Methods of Digital-to-Analog Conversion

16. In the 4-bit DAC in Figure 11–26, the lowest-weighted resistor has a value of 10 kΩ. What should the values of the other input resistors be?

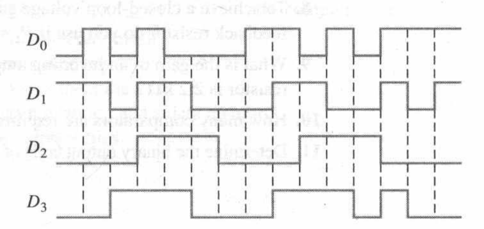
17. Determine the output of the DAC in Figure 11–40(a) if the sequence of 4-bit numbers in part (b) is applied to the inputs. The data inputs have a low value of 0 V and a high value of +5 V.

▶ FIGURE 11–40



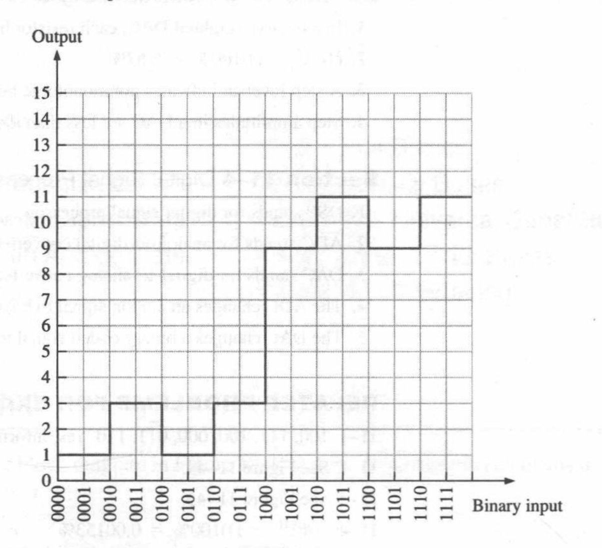
18. Repeat Problem 17 for the inputs in Figure 11–41.

▶ FIGURE 11–41



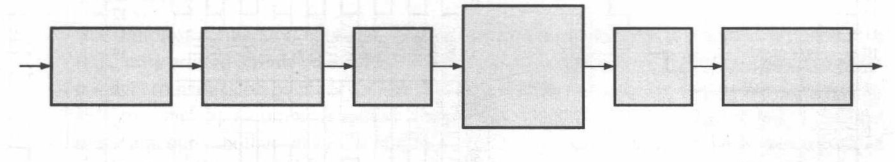
19. Determine the resolution expressed as a percentage, for each of the following DACs:
    (a) 3-bit (b) 10-bit (c) 18-bit
20. Develop a circuit for generating an 8-bit binary test sequence for the test setup in Figure 11–31.
21. A 4-bit DAC has failed in such a way that the MSB is stuck in the 0 state. Draw the analog output when a straight binary sequence is applied to the inputs.
22. A straight binary sequence is applied to a 4-bit DAC, and the output in Figure 11–42 is observed. What is the problem?

► FIGURE 11–42



Section 11–4 Digital Signal Processing

23. Explain the purpose of analog-to-digital conversion.
24. Fill in the appropriate functional names for the digital signal processing system block diagram in Figure 11–43.



▲ FIGURE 11–43

25. Explain the purpose of digital-to-analog conversion.

ANSWERS

SECTION CHECKUPS

Section 11–1 Analog-to-Digital Conversion

1. Sampling is the process of converting an analog signal into a series of impulses, each representing the amplitude of the analog signal.
2. A sampled value is held to allow time to convert the value to a binary code.
3. The minimum sampling frequency is 40 kHz.
4. Quantization is the process of converting a sampled level to a binary code.
5. The number of bits determine quantization accuracy.

Section 11–2 Methods of Analog-to-Digital Conversion

1. The simultaneous (flash) method is fastest.
2. The sigma-delta method produces a single-bit data stream.
3. Yes, successive approximation has a fixed conversion time.
4. Missing code, incorrect code, and offset are types of ADC output errors.

Section 11–3 Methods of Digital-to-Analog Conversion

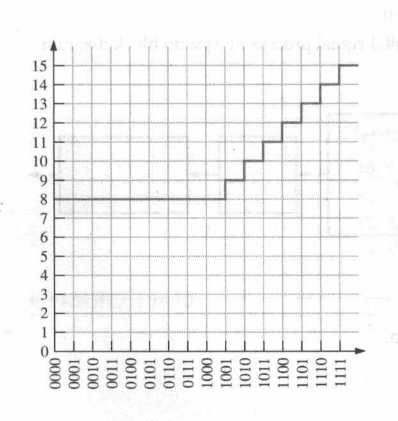
1. In a binary-weighted DAC, each resistor has a different value.
2. $(1/(2^4 - 1))100\% = 6.67\%$
3. A step reversal indicates nonmonotonic behavior in a DAC.
4. Step amplitudes in a DAC are less than ideal with low gain.

Section 11–4 Digital Signal Processing

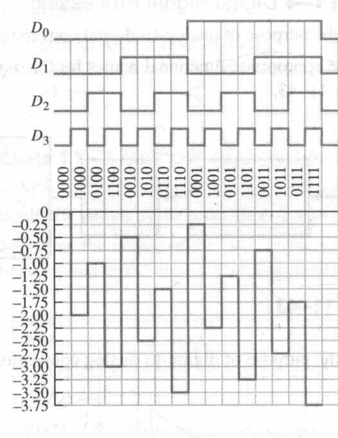
1. DSP stands for digital signal processor.
2. ADC stands for analog-to-digital converter.
3. DAC stands for digital-to-analog converter.
4. The ADC changes an analog signal to binary coded form.
5. The DAC changes a binary coded signal to analog form.

RELATED PROBLEMS FOR EXAMPLES

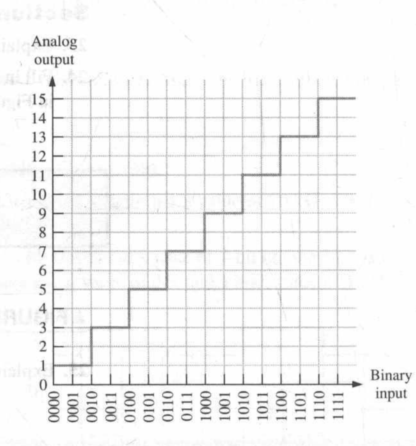
11–1 100, 111, 100, 000, 011, 110. Yes, information is lost.
11–2 See Figure 11–44.
11–3 See Figure 11–45.
11–4 $(1/(2^{16} - 1))100\% = 0.00153\%$
11–5 See Figure 11–46.



▲ FIGURE 11–44



▲ FIGURE 11–45



▲ FIGURE 11–46

TRUE/FALSE QUIZ

1. T **2.** F **3.** F **4.** T **5.** T **6.** F **7.** T **8.** T **9.** T **10.** F

SELF-TEST

1. (b) **2.** (d) **3.** (a) **4.** (b) **5.** (d) **6.** (a)
7. (e) **8.** (b) **9.** (e) **10.** (c) **11.** (b) **12.** (c)
13. (a) **14.** (c) **15.** (a) **16.** (c) **17.** (a) **18.** (d)

Chapter 12 Data Transmission

CHAPTER OUTLINE

12-1 Data Transmission Media

InfoNote

Early fundamental work in data transmission and information theory was done by Harry Nyquist, Ralph Hartley, Claude Shannon, and others.

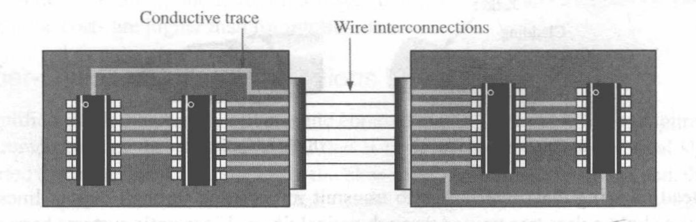
The basic block diagram in Figure 12–1 illustrates the essential elements in a data transmission system.

Sending device	Transmission medium	Receiving device

▶ **FIGURE 12–1**
Basic data transmission system.

Wire Connections

The simplest connection between sending and receiving devices is a wire or a conductive trace on a printed circuit board (PCB). This type of connection is typically limited to internal data transmission over very short distances within the same system or between nearby systems, such as a computer and/or peripherals. Data buses or conductive traces connect one element to another on a PCB and between PCBs in close proximity or between parts of a system, as illustrated in Figure 12–2.

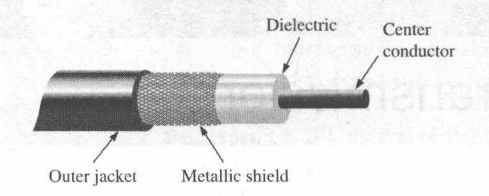


▶ **FIGURE 12–2**
Conductive traces on PCBs and wire interconnections between boards.

Coaxial Cable

Coaxial cable (coax) consists of a center conductor within an insulating dielectric material. A copper braided or foil shield surrounds the dielectric to protect the conductor against electromagnetic interference (EMI). The shield is encased in a protective insulating jacket, as shown in Figure 12–3. BNC (Bayonet Neill-Concelman) connectors are typically used

for coaxial connections. Coax is used in data transmission applications with data rates up to about 1 GHz. Two common applications for coax are cable TV and Internet connections.

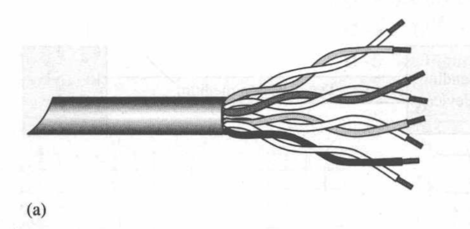


◀ **FIGURE 12–3**
Construction view of a coaxial cable.

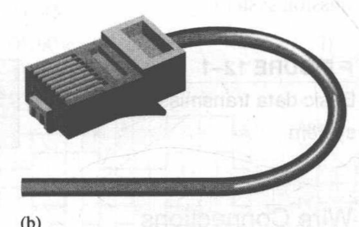
Twisted Pair Cable

Unshielded twisted pair (**UTP**) cable is used extensively for indoor telephone application as well as some outdoor uses. It is found in many computer networks and video applications, such as security cameras, and also in the USB (universal serial bus) cable. UTP is color-coded according to a standard 25-pair color code. Most cables use a subset of these standard colors.

Cross talk, a type of distortion, is minimized when twisted pairs are bundled together. The two wires in each pair are twisted so that they cross each other at nearly 90°, ideally cancelling any electromagnetic fields generated by the signals in the wires. UTP cables are limited to use in low-noise environments and to lower signal frequencies than coax, such as audio and other signals up to about 1 MHz. UTP cables use standard RJ-45 connectors. A common four-pair UTP cable is shown in Figure 12–4(a), and an RJ-45 connector is shown in part (b). Shielded twisted pair (**STP**) cable encased in a metal sleeve or conduit is also available and provides more protection from EMI.



(a)

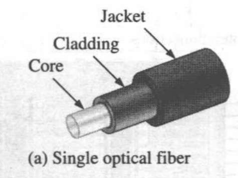


(b)

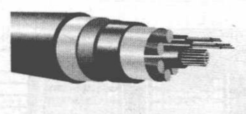
◀ **FIGURE 12–4**
Example of an unshielded twisted pair (UTP) cable and connector.

Optical Fiber Cable

The structure of a single **optical fiber** is shown in Figure 12–5(a). An optical fiber can be as small as a human hair, so many single fibers can be bundled into a cable, as shown in Figure 12–5(b).



(a) Single optical fiber



(b) Fiber optic cable

◀ **FIGURE 12–5**
Optical fiber cables.

Instead of using electrical pulses to transmit information through copper lines, fiber optics uses light pulses transmitted through optical fibers. Fiber-optic systems have several advantages over electrical transmission media. Advantages include faster data rates, higher signal capacity (more signals at a time), and better transmission over longer distances; optical fibers are not susceptible to EMI. The main disadvantage of fiber optics is the cost, which is higher than that of coax, UTP, and STP.

Optical fiber is commonly used as a medium for telecommunication and networking. Because light propagates through the fiber with little attenuation compared to electrical cables, optical fiber is useful for long-distance transmission. Data rates from 10 GHz to 40 GHz are common, although rates over 100 GHz are used.

When light is introduced at one end of an optical fiber called the *core*, it "bounces" along until it emerges from the other end, as shown in Figure 12–6. The fiber is typically made of pure glass, plastic, or other material that is surrounded by a highly reflective cladding that effectively acts as a mirrored surface, using a phenomenon called *total internal reflection* to produce an almost lossless reflection. This allows the light to move around bends in the fiber.

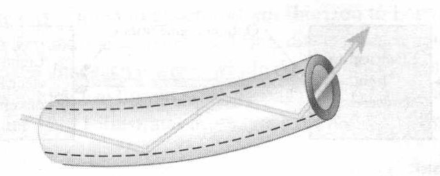


FIGURE 12–6
Light propagating through an optical fiber while reflecting off the internal surface.

Modes of Light Propagation

Two basic modes of light propagation in optical fibers are multimode and single-mode, as illustrated in Figure 12–7. In **multimode**, the light entering the fiber will tend to propagate through the core in multiple rays (modes), basically due to varying angles as each light ray moves along. Some of the rays will go straight down the core, while others will bounce back and forth. Still others will scatter due to the sharp angle at which they strike the cladding, resulting in attenuation in light energy. Multimode also exhibits time dispersion, which means that all the light rays do not arrive at the end of the fiber at exactly the same time. In **single-mode**, the core is much smaller in diameter than in multimode. Light entering the fiber tends to propagate in a straight line as a single ray.

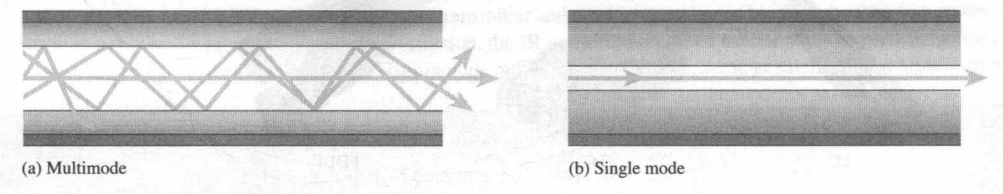


FIGURE 12–7
Modes of light propagation in an optical fiber.

The diameter of the optical fiber determines the mode. There are three sizes most widely used in data transmission: 50/125, 62.5/125 and 8.3/125. The numbers are in microns (one micron is one millionth of a meter) and represent the diameters of the fiber core and cladding, respectively. The 50/125 and the 62.5/125 are multimode fibers. The 8.3/125 is a single-mode fiber. Single mode results in increased bandwidth and distance for transmission, but the costs are higher than for multimode.

A Fiber-Optic Data Communications Link

A simplified block diagram of a fiber-optic communications link is shown in Figure 12–8. The source provides the electrical signal that is to be transmitted. The electrical signal is converted to a light signal and coupled to the fiber-optic cable. At the receiving end, the light signal is coupled out of the cable into the receiver, which converts it back to an electrical signal. The signal is then processed and sent to the end user. The electrical signal modulates the light intensity and produces a light signal that carries the same information as the electrical signal. Special connectors are used to connect the fiber-optic cable to various equipment.

Various types of connectors are used in fiber-optic systems. Some of these are described as follows and are shown in Figure 12–9:

- ST An AT&T trademark and is one of the most widely used for multimode networks
- SC A snap-in type multimode connector
- FC A popular single-mode connector type

- LC A single-mode connector
- LX-5 Similar to an LC connector except it has a shutter over the end of the fiber
- MT A 12-fiber connector used for ribbon cables
- FDDI All duplex, meaning the connector can accommodate two optical fibers for two-way communication

Electrical signal source → Electrical-to-light conversion → Optical connectors / Fiber-optic cable → Light-to-electrical conversion → Receiver

FIGURE 12–8 Basic block diagram of a fiber-optic communications link.

ST SC FC

LC MT FDDI

FIGURE 12–9 Typical types of optical fiber connectors.

Wireless Transmission

The transmission of data through air and space via electromagnetic waves without the use of physical connections between sending and receiving systems is known as *wireless transmission*. Wireless transmission generally can be categorized by the type of signal in terms of application, frequency, or how the data are configured. Another medium of wireless communications is water where sonar is used. Sonar produces low-frequency sound waves that do not fall into the electromagnetic spectrum.

The Electromagnetic Spectrum

The spectrum of frequencies for the range of **electromagnetic waves** is shown in Figure 12–10. Most data communications occur within the radio wave, microwave, and infrared frequencies.

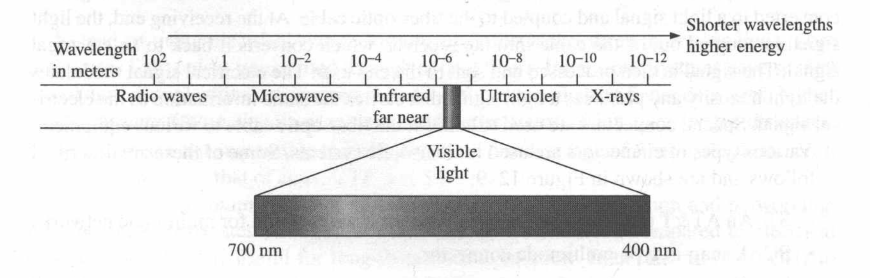
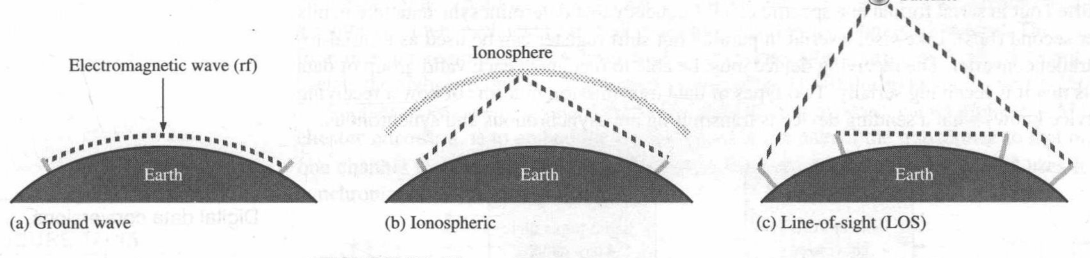


FIGURE 12–10 The electromagnetic spectrum.

Three ways in which radio wave (rf) and microwave signals propagate through Earth's atmosphere (air) are ground wave, ionospheric, and line-of-sight. In ground wave propagation, the radio waves follow the curvature of Earth and can be up to about 2 MHz in frequency; the standard AM broadcast band is an example. Radio frequencies in the 30 MHz to 85 MHz range bounce off of the ionosphere. These signals can change with time of day and weather conditions. Most ham radio bands are examples of where signals bounce off the ionosphere. In line-of-sight (LOS) propagation, the receiver must be in view of the transmitter. The distance is limited to about 100 km (horizon to horizon) from a ground-based transmitter to a ground-based receiver. Long distances are achieved by placing a series of repeater towers so that each tower is within the line-of-sight of the previous tower. In the case of satellites, which use line-of-sight propagation, the distances can be extended around the world. Figure 12–11 illustrates these types of rf and microwave propagation.



▲ **FIGURE 12–11**

Ways in which rf and microwave signals can propagate.

Communication in the infrared region of the electromagnetic spectrum can be line-of-sight or diffused. With LOS, the transmitter and receiver must be visible to each other with no obstacles in between. With diffusion, the IR waves reflect off of nearby surfaces such as buildings, ceilings, and walls. Uses include remote control devices, weather satellites, and night vision.

SECTION 12-1 CHECKUP

Answers are at the end of the chapter.

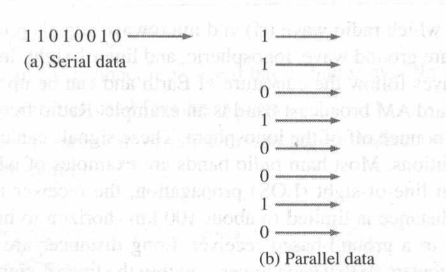
1. List the types of data transmission media.
2. What is the purpose of a coax shield?
3. Name three ways in which radio waves are propagated in wireless transmission.
4. Which type of electromagnetic radiation has the highest frequencies?
5. Generally, what is the difference between baseband and broadband?

12-2 Methods and Modes of Data Transmission

Serial and Parallel Data

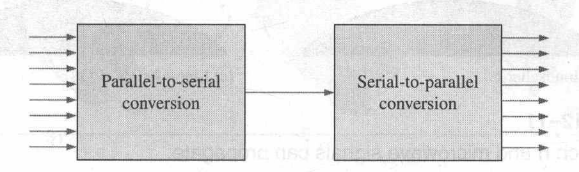
Serial data transmission is when data are transmitted one bit at a time in a bit stream, as illustrated in Figure 12–12(a). Parallel data transmission is when data are transmitted several bits at a time, as shown in part (b). In general, a given number of bits can be transmitted faster in parallel than in series, resulting in higher data rates. However, when several bits are sent simultaneously on separate lines in parallel, slight differences in the properties of the lines can cause skewing in the data, making the data more susceptible to error, so the data rate may need to be reduced to prevent errors. Error detection and correction methods can be used in these cases.

Generally, data are processed in parallel by computers but are transmitted serially to outside systems. For example, data from a computer to a printer are typically sent over a USB, which is serial. Data that are sent over long distances via one of the transmission media are typically in a serial format. In some cases, data can be sent in parallel over a single channel by using different frequencies for each bit, so the bits can be transmitted at the same time.



▲FIGURE 12–12

Serial-to-parallel and parallel-to-serial conversions are used in most data transmission systems. The basic concept is shown in Figure 12–13. A simple parallel-to-serial converter is a parallel in/serial out shift register. The parallel bits are loaded into the register and then shifted out in serial format at a specific clock frequency that determines the data rate in bits per second (bps). Likewise, a serial in/parallel out shift register can be used as a serial-to-parallel converter. The receiving device must be able to recognize each valid group of data bits that it is receiving serially. Two types of data transmission in terms of how a receiving device knows what a sending device is transmitting are asynchronous and synchronous.



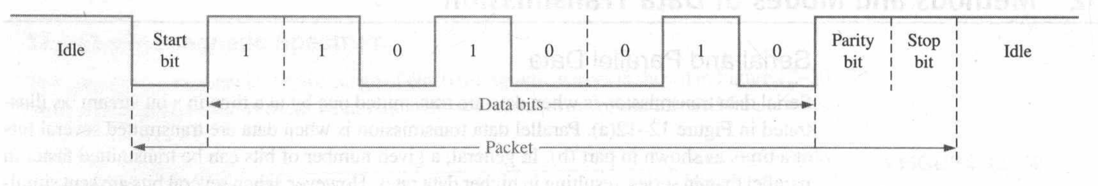
◀FIGURE 12–13

Digital data conversions.

Asynchronous Data

Data are sent in short "bursts" known as packets in asynchronous transmission. A data **packet** is one complete piece of information of a longer message. Typically, many packets make up the entire message. A data packet consists of data bits representing alphabetic or numeric characters, a parity bit, and start/stop bits. There is a pause between data packets so that the receiver recognizes the start bit that precedes each packet. At the end of the data packet, there are one or more stop bits that tell the receiver the packet is complete.

In **asynchronous** systems, the sending and receiving devices operate with separate oscillators having the same clock frequency. Because the separate clock frequencies may drift over time, they are typically re-synchronized on each data packet with the start bit. Most commonly, data are sent in small packets of perhaps 10 or 11 bits. Eight of these bits carry the information. Between packets, when the channel is idle, there is a continuous logic level. A data packet always begins with a start bit with the opposite logic level as the idle period to alert the receiver that a data packet is starting. A parity bit follows the eight data bits, and a stop bit signals the end of the packet. This is illustrated in Figure 12–14.



▲FIGURE 12–14

Example of a serial transmission of a data packet for a given data code.

Synchronous Data

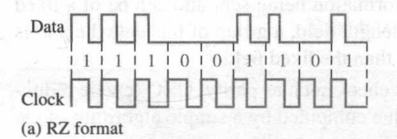
In **synchronous** data transmission, both the sender and the receiver derive timing from the same clock signal, which originates at the sender end of the system. The bits are transmitted in a continuous stream with no pauses, so the receiver must have some way to recognize where a data block starts and ends. In order for the receiver to know when to read infor-

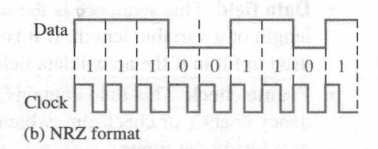
mation bits from the channel, it must determine exactly when the data begin and the time between bits. When this timing information is determined, the receiver is synchronized with the transmitter. Unlike asynchronous transmission, the data blocks usually contain more than one character of information. Synchronous transmission is generally faster than asynchronous transmission.

One method of synchronization is by using separate channels to transmit the data and the timing information (synchronization and clock pulses). Because the transmitter originates both the data and the timing pulses, the receiver will read the data channel only when told to do so by the transmitter (via the timing channel), and synchronization is achieved. The disadvantage of this method is that it requires two physical lines.

Two data formats that require separate data and timing are **RZ** (return to zero) and **NRZ** (nonreturn to zero). In the RZ format, a single pulse during a bit time represents a 1 and the absence of a pulse is a zero, as shown in Figure 12–15(a). In the NRZ format, a high level during a bit time represents a 1 and a low level represents a 0. A series of 1s is represented by a continuous high level, and a series of 0s is represented by a continuous low level. The waveform does not return to the low level until a zero occurs after a string of 1s and does not go back to the high level until a 1 occurs after a string of 0s. This is illustrated in Figure 12–15(b).

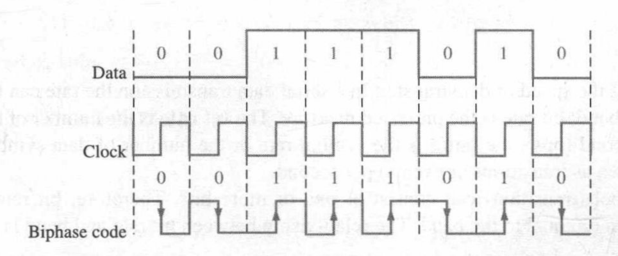
Another more commonly used method of data synchronization, called biphase or **Manchester encoding**, is to embed the timing signal in the data at the transmitter so that only one channel is required. The receiver extracts the embedded timing signal and uses it to synchronize to the transmitter.





▶ **FIGURE 12–15**
Data formats that require separate timing for synchronization.

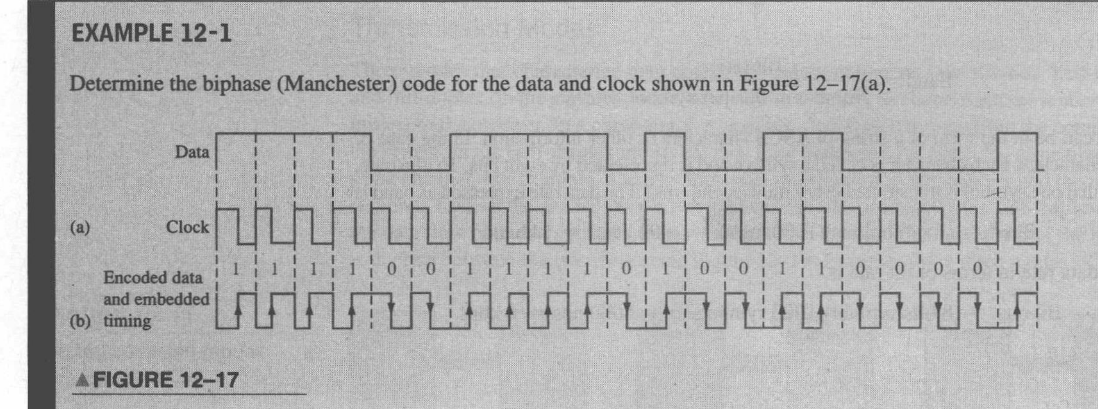
Figure 12–16 illustrates Manchester encoding. A rising edge in the biphase code is a 1 and a falling edge is a 0, as indicated by the up and down arrowheads. The edges occur at the middle of the bit time. The biphase code is sent to the receiver, and the clock is extracted from the data with a phase-locked loop. Sometimes a series of all 1s or all 0s are included in the transmission to allow the receiver to synchronize.



▶ **FIGURE 12–16**
Example of Manchester encoded data and timing.

EXAMPLE 12-1

Determine the biphase (Manchester) code for the data and clock shown in Figure 12–17(a).



▲ **FIGURE 12–17**

Solution

The encoded data and embedded timing are shown in Figure 12–17(b). As the arrowheads indicate, the rising edges are 1s and the falling edges are 0s that occur in the middle of each bit time (period of the clock).

Related Problem*

If the data were all 1s, what would the Manchester code look like?

*Answers are at the end of the chapter.

Synchronous Frames

Synchronous data are sent in frames that include other bits, as shown by the generic protocol in Figure 12–18. (Frame formats vary because there are numerous standards in use.)

- **Preamble** A group of bits at the beginning of a frame that is used to alert the receiver that a new frame has arrived and to synchronize the receiver's clock with the transmitted clock
- **Address fields** A group of bits containing the address(s) of the sender and the receiver. One or both addresses may be present in a given protocol.
- **Control field** This group of bits identifies the type of data being sent, such as handshaking (establishes a connection), file transfers, and the size of the data.
- **Data field** This sequence is the actual information being sent and can be of a fixed length or a variable length. If it is a fixed-length field, a group of bits called a *pad* is used to fill in if the actual data field is less than the fixed field.
- **Frame check** This field contains an error check such as parity, CRC (cyclic redundancy check), or checksum, which is a value computed by a simple algorithm of the data bits in the frame.
- **End frame** A group of bits that tells the receiver when the end of the frame occurs.

Preamble	Address fields	Control field	Data	Frame check	End frame

FIGURE 12–18 Basic synchronous frame structure.

Data Rate

Data rate is the speed of data transfer. In a serial data transmission the rate can be stated as bit rate or baud; bit rate is the preferred measure. The **bit rate** is the number of bits (1s and 0s) per second (bps); the **baud** is the symbol rate or the number of data symbols (sometimes known as transitions or events) per second.

A symbol (transition) can consist of one or more bits. Therefore, bit rate is always greater than or equal to the baud. The relationship between bit rate and baud is

$$\text{Bit rate} = (\text{Number of bits per symbol})(\text{Baud})$$

or

$$\text{Baud} = \frac{\text{Bit rate}}{\text{Number of bits per symbol}}$$

Data can be in the form of a string of ASCII characters or other information. In the case of ASCII characters, each character is called a symbol and is represented by eight bits. To illustrate, assume that one symbol is transmitted every millisecond (ms). The data rate expressed as baud is

$$\text{Baud} = (1\ \text{symbol/ms})(1000\ \text{ms/s}) = 1000\ \text{baud} = 1\ \text{kbaud}$$

The data rate in terms of bit rate is

$$\text{Bit rate} = (8\ \text{bits/symbol})(1000\ \text{symbols/s}) = 8000\ \text{bps} = 8\ \text{kbps}$$

EXAMPLE 12-2

A certain analog waveform is represented by sixteen-voltage levels that are being transmitted. Each level (symbol) is represented by a 4-bit code. Assuming that eight symbols are transmitted in 1 μs, express the data rate as bit rate and as baud.

Solution

$$\text{Bit rate} = (4\ \text{bits/symbol})(8\ \text{symbols}/\mu\text{s}) = 32\ \text{bits}/\mu\text{s} = \mathbf{32\ Mbps}$$

$$\text{Baud} = \frac{32\ \text{Mbps}}{4\ \text{bits per symbol}} = \mathbf{8\ Mbaud}$$

Related Problem

Determine the bit rate if a symbol is represented by 8 bits and the baud is 5000 symbols/s.

Transmission Efficiency

The efficiency of a data transmission channel is the ratio of data bits to total bits in a packet. For example, in Figure 12–14 there are eight data bits, a start bit, a parity bit, and a stop bit. The nondata bits are considered overhead bits. There are eleven total bits transmitted in a packet so the efficiency of the transmission is

$$\text{Efficiency} = \frac{\text{Data bits}}{\text{Total bits}} = \frac{8\ \text{bits}}{11\ \text{bits}} = 0.727 \text{ or } 72.7\%$$

EXAMPLE 12-3

A certain system transmits a block of information containing ten packets each with eight data bits, a start bit, and a stop bit. Additional "overhead" bits include a 4-bit synchronization code at the beginning of the block and a parity bit at the end of the block. Determine the transmission efficiency.

Solution

$$\text{Data bits} = (8\ \text{data bits})(10\ \text{packets}) = 80\ \text{bits}$$

$$\text{Overhead bits} = (1)(10\ \text{start bits}) + (1)(10\ \text{stop bits}) + 4\ \text{synchronization bits} + 1\ \text{parity bit} = 25\ \text{bits}$$

$$\text{Total bits} = \text{Data bits} + \text{Overhead bits} = 80 + 25 = 105$$

$$\text{Efficiency} = \frac{\text{Data bits}}{\text{Total bits}} = \frac{80}{105} = \mathbf{0.762 \text{ or } 76.2\%}$$

Related Problem

Determine the efficiency if each packet has 12 data bits and the same number of overhead bits as stated in the example.

Transmission Modes

Three modes that characterize data channel (media) connections are simplex, half-duplex, and full-duplex. In the **simplex** mode, data flow in only one direction from the sender (transmitter) to the receiver. In a computer, for example, data flow one way from the computer to the printer. In the **half-duplex** mode, the data flow both ways but not at the same time in the same channel. For example, a sender transmits information to the receiver and the receiver responds back to the sender after it has received the information. In the **full-duplex** mode, the data flow both ways simultaneously in the same channel. The bandwidth of the channel is divided between the two directions. Figure 12–19 illustrates these three modes.

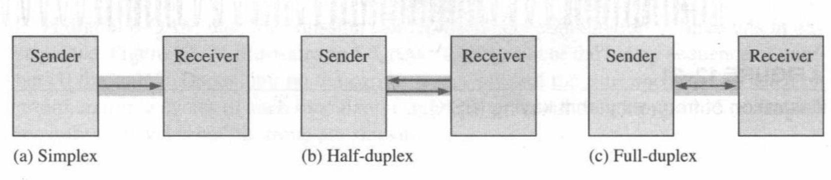


FIGURE 12–19
Data transmission modes.

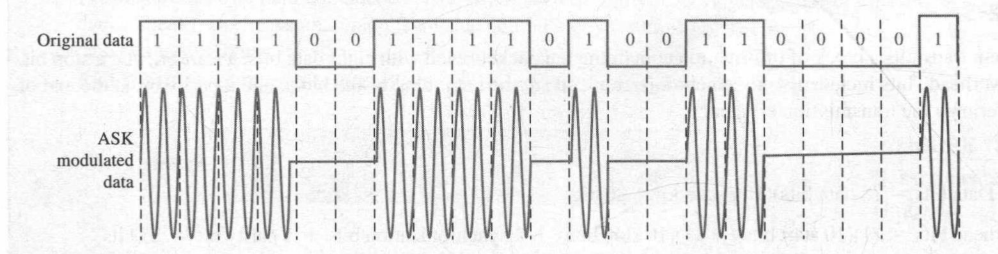
SECTION 12-2 CHECKUP

1. Explain the difference between serial and parallel data.
2. What is the purpose of synchronization in a data transmission system?
3. Name three types of data formats.
4. List the modes of data transmission?

12-3 Modulation of Analog Signals with Digital Data

Amplitude-Shift Keying

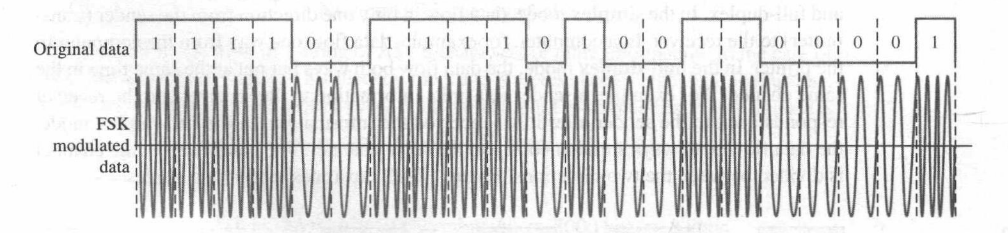
Amplitude-shift keying **(ASK)** is a form of **modulation** in which a digital signal varies the amplitude of a higher frequency sine wave (carrier). In its simplest form, a sinusoidal carrier signal is turned on and off by the data signal and, therefore, this method is also known as on-off keying (OOK). When the carrier is *on*, a binary 1 is represented, and when the carrier is *off*, a binary 0 is represented. ASK is very susceptible to noise interference and is not typically used for wireless data transmission. ASK is most commonly used in fiber optics where the presence of light represents a binary 1 and the absence of light represents a binary 0. Figure 12–20 illustrates the concept of ASK. The presence of the sine wave carrier is a 1 and the absence is a 0. When modulated by digital data (1s and 0s), this method is sometimes known as binary amplitude-shift keying (BASK).



▲**FIGURE 12–20**
Illustration of amplitude-shift keying (ASK).

Frequency-Shift Keying

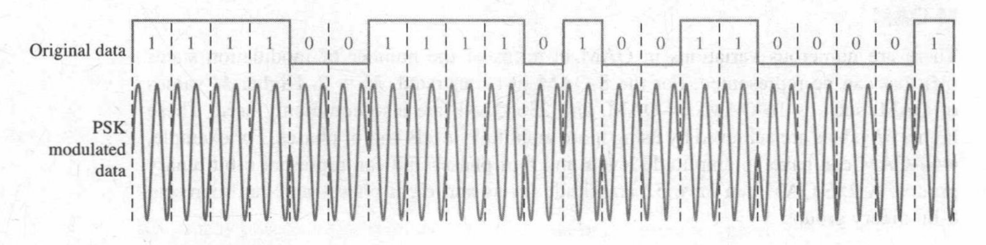
Frequency-shift keying **(FSK)** is a form of modulation in which a digital signal modulates the frequency of a higher frequency sine wave (carrier). A carrier signal with a lower frequency generally represents a binary 0, and a carrier signal with a higher frequency represents a binary 1. When modulated by digital data (1s and 0s), this method is sometimes known as binary frequency-shift keying (BFSK). Figure 12–21 illustrates FSK.



▲**FIGURE 12–21**
Illustration of frequency-shift keying (FSK).

Phase-Shift Keying

Phase-shift keying **(PSK)** is a form of modulation in which a digital signal modulates the phase of a higher frequency sine wave. A carrier signal of one phase generally represents a binary 1, and a carrier signal that is 180° out-of-phase represents a binary 0. When modulated by digital data (1s and 0s), this method is sometimes known as binary phase-shift keying (BPSK). In one of its many variations, PSK applications include wireless LAN and bluetooth. Figure 12–22 illustrates PSK.

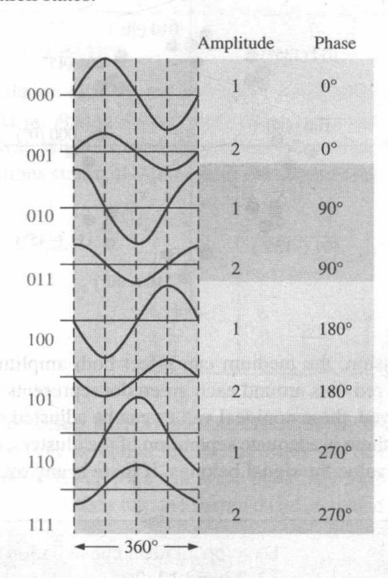


▲ **FIGURE 12–22**

Illustration of phase-shift keying (PSK).

Quadrature Amplitude Modulation

Quadrature amplitude modulation **(QAM)** is widely used in telecommunications and in digital cable TV. Digital QAM uses a combination of PSK and ASK to send information. Quadrature refers to a 90° phase difference. Each combination of phase and amplitude is called a *modulation state* or *symbol* and represents a combination of two or more bits. To illustrate the basic concept of QAM, let's use what is known as 8-QAM where each of the eight modulation states (2^3) represents a unique three-bit combination. As shown in Figure 12–23, in 8-QAM, there are two different amplitudes with a quadrature phase difference between each pair. Since there are four quadratures (90°) in 360°, there are a total of eight different amplitude/phase combinations or modulation states.



► **FIGURE 12–23**

Eight amplitude/phase combinations (modulation states) represent one of the eight 3-bit groups. Only one cycle of each modulation state is shown.

A digital 8-QAM data transmission can represent any combination of three bits in any sequence. Figure 12–24 illustrates an 8-QAM transmission of the binary sequence of numbers 0 through 7. Depending on the carrier frequency and the time specified for each bit group, multiple cycles of each modulation state can represent each bit group. For simplicity, only two cycles per bit group are shown.

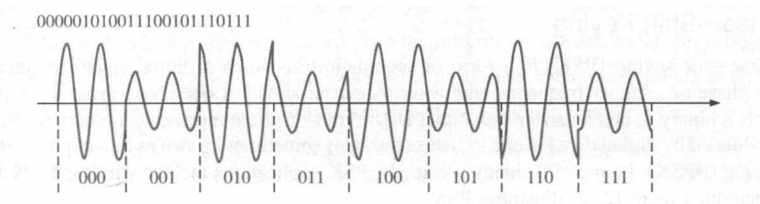


FIGURE 12–24 Illustration of an 8-QAM transmission of the binary sequence shown.

M-QAM

There are numerous variations in QAM in terms of the number of modulation states (M) that can be represented. For the 8-QAM just illustrated, $M = 8$. Higher M values of QAM, such as 16-QAM, 64-QAM, and 256-QAM, are also commonly used. These higher M values are achieved by using more amplitude levels and/or phases. For example, a 64-QAM can have 16 amplitude levels and four phases and can represent 6-bit binary groups. A 256-QAM can have 32 amplitude levels and eight phases and can represent 8-bit binary groups.

Constellation Maps

Modulated transmission of digital data can be represented by a constellation map, which is a vector representation that graphically shows the symbol values and corresponding phases being transmitted by a system. As you have seen, when data are transmitted, a pattern is modulated into the signal, such as in PSK, where the bit pattern is represented by various phase shifts. A constellation map is useful in the design and analysis of a data transmission system and in visually understanding how the system works.

Figure 12–25 shows a 4-quadrant constellation map for a 3-bit PSK transmission. Each green dot represents the ideal amplitude and phase of the modulated signal. The amplitude constant is represented by the distance of the dots from the origin. Eight different phases or symbols represent the 3-bit binary combinations.

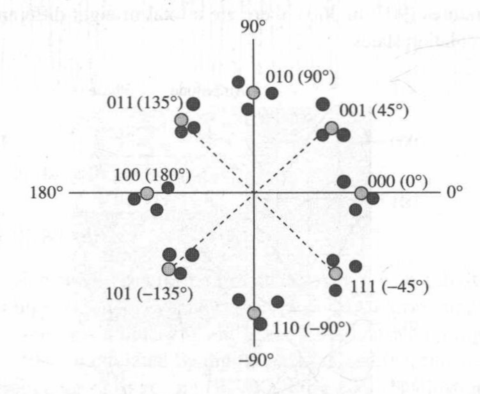


FIGURE 12–25 Constellation map for a 3-bit PSK transmission. The phases are 0°, 45°, 90°, 135°, 180°, −45°, −90°, and −135°, as indicated.

In an actual transmission, the medium can affect both amplitude and phase shift. In the figure, the cluster of red dots around each green dot represents nonideal signal values. When the signal is received, these nonideal values can be adjusted to the ideal value (nearest green dot) as long as there is adequate separation of the clusters; there should be no confusion as to which ideal value the signal belongs. If there is any overlap, errors can occur.

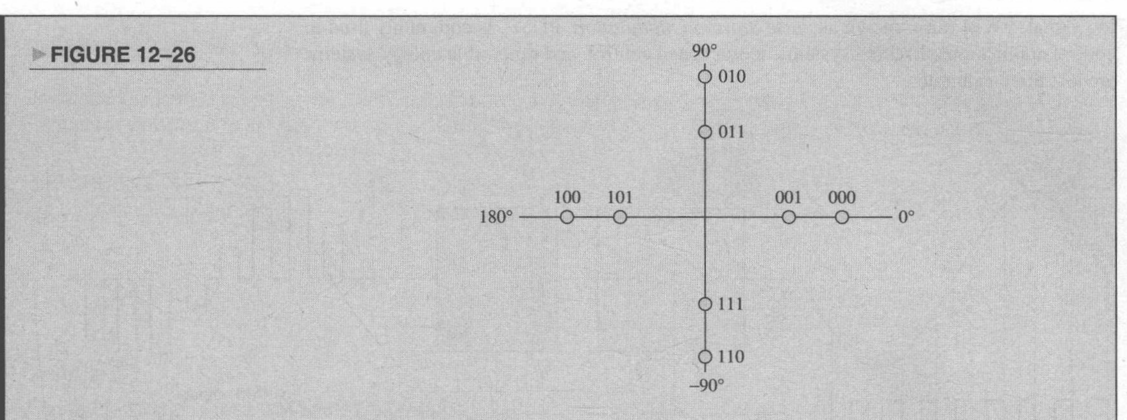
EXAMPLE 12-4

Develop an ideal constellation map for the 8-QAM transmission represented in Figures 12–23 and 12–24.

Solution

There are four phases and four amplitudes that represent a 3-bit code in this system. The ideal constellation map is shown in Figure 12–26. There are two amplitudes for each phase represented by the distance from the origin.

FIGURE 12–26



Related Problem

How many phases and amplitudes could be used to represent a 4-bit code?

SECTION 12-3 CHECKUP

1. List four types of modulation techniques.
2. What parameter is changed in ASK?
3. What parameter is changed in FSK?
4. What is QAM?
5. What parameter is changed in PSK?

12-4 Modulation of Digital Signals with Analog Data

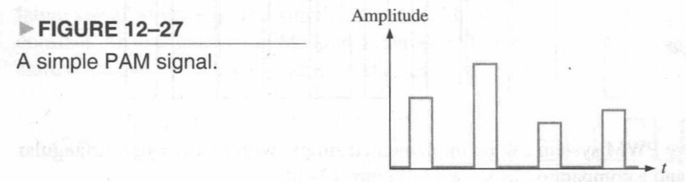
Pulse Amplitude Modulation

In pulse amplitude modulation (**PAM**), the heights or amplitudes of the pulses are varied according to the modulating analog signal; each pulse represents a value of the analog signal. PAM is the simplest, but least used, type of pulse modulation although it is used in the Ethernet communications standard. A simple PAM sequence is shown in Figure 12–27.

InfoNote

Ethernet is a family of computer networking protocols described by the IEEE 802.3 standard. Systems that communicate using Ethernet divide the data into individual packets called frames. Each frame contains source and destination addresses and error-checking bits. The Ethernet standard includes several variations that specify both media and signaling standards, including type of wire or cable, data format, and data rates.

FIGURE 12–27
A simple PAM signal.

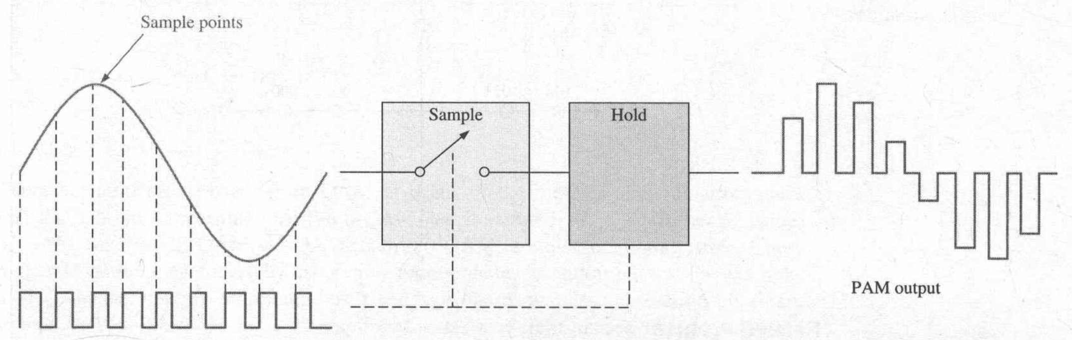


A basic method of producing a PAM representation of an analog signal is to use a constant-amplitude pulse source to sample the analog wave that has a frequency lower than the pulses, as shown in Figure 12–28 for a sine wave input; any form of analog signal can be converted to a PAM output. The pulses turn the switch on (closed) and off (open) to sample the waveform. When there is a pulse, the sample switch is closed; the amplitude of the sine wave at that point goes to the hold element that maintains the initial analog value occurring at the beginning of each pulse for the duration of the pulse. The output goes to zero between pulses.

Pulse Width Modulation

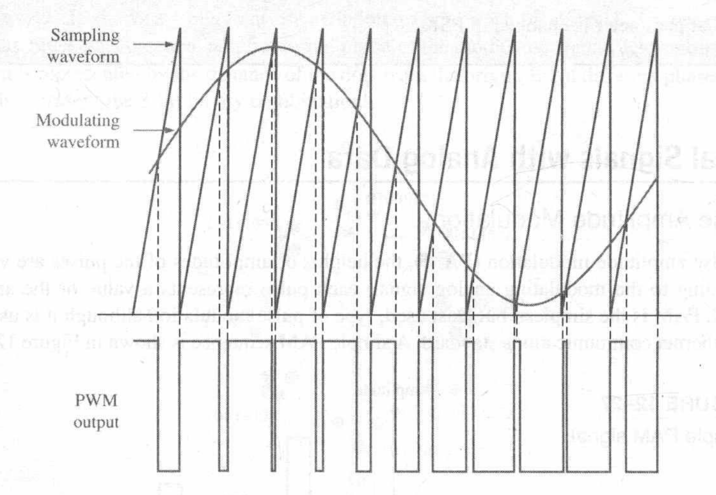
In pulse width modulation (**PWM**), the width or duration of the pulses and duty cycle are varied according to the modulating analog signal; each pulse represents a value of the ana-

log signal. PWM (also known as *pulse duration modulation*, PDM), is commonly used in control applications. Braking systems, motor speed control, and renewable energy systems are just three examples.



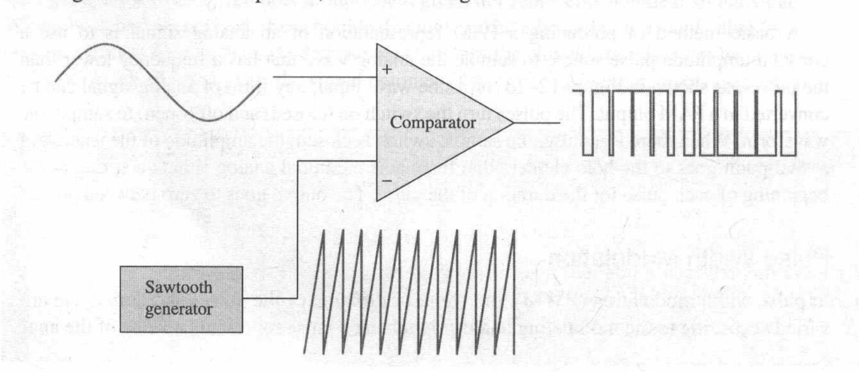
▲ **FIGURE 12–28**
Basic method of pulse amplitude modulation.

Figure 12–29 illustrates one method of PWM generation, called the *intersective method*, which uses a sawtooth waveform. A triangular waveform can also be used. Again, a sine wave input is used, but the input can be any type of analog waveform. The sawtooth intersects the sinusoidal modulating signal twice during each cycle. The sawtooth is either increasing above the sine wave or decreasing below the sine wave. When the sawtooth is increasing above the sine wave, a low level is generated; when it is decreasing below the sine wave, a high level is generated. The resulting output is a series of pulses with widths proportional to the amplitude of the sine wave.



◀ **FIGURE 12–29**
Illustration of PWM.

An intersective PWM system can be implemented simply with a sawtooth or triangular wave generator and a comparator, as shown in Figure 12–30.

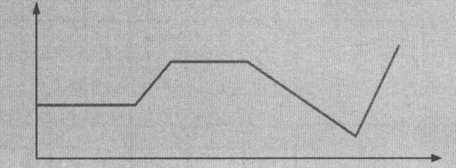


◀ **FIGURE 12–30**
A basic method of pulse width modulation.

EXAMPLE 12-5

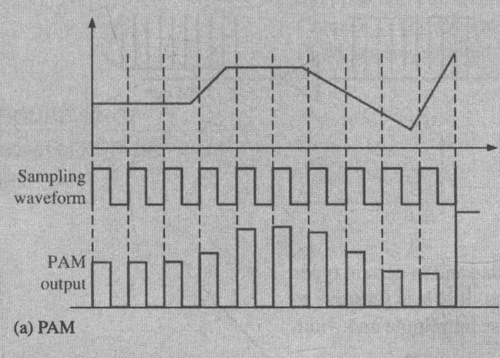
Determine the PAM signal and the PWM signal for the modulating signal in Figure 12–31. Assume ten cycles of the sampling pulse waveform or sawtooth waveform during the portion of the modulating signal shown.

▶ **FIGURE 12–31**
Modulating waveform.

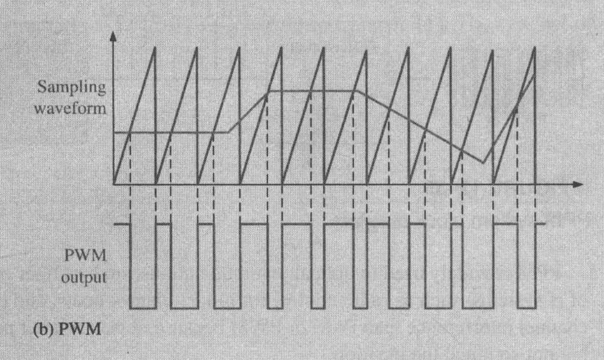


Solution

Figure 12–32 shows the PAM and PWM results.



(a) PAM



(b) PWM

▲ **FIGURE 12–32**

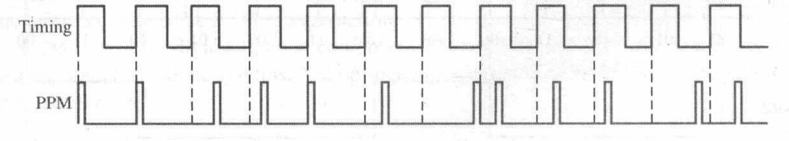
Related Problem

How would the outputs in Figure 12–32 be affected by an increase in frequency of the pulse and sawtooth waveforms?

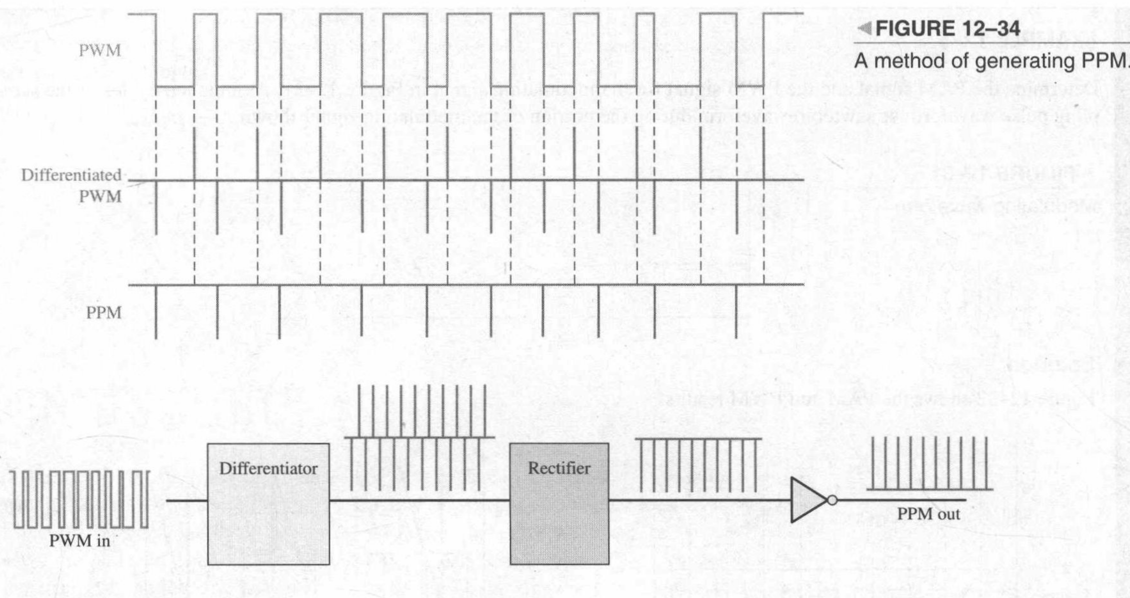
Pulse Position Modulation

In pulse position modulation **(PPM)**, also known as pulse phase modulation, the position of each pulse relative to a reference or timing signal is varied proportional to the modulating signal waveform. The amplitude and width of the pulses in a PPM system are kept constant. An example of a PPM signal is shown in Figure 12–33 where the PPM pulses are shifted relative to the leading edges of the timing waveform.

▶ **FIGURE 12–33**
Example of a PPM signal with timing.



As with other types of pulse modulation, there is generally more than one way to produce a modulated waveform. One method is to derive the PPM from PWM, as illustrated in Figure 12–34. Notice that the leading (positive-going) edges of the PWM signal occur at fixed intervals, while the trailing (negative-going) edges vary relative to the leading edges. When the PWM signal is passed through a differentiator, short positive pulses (spikes) are generated on the leading edges and short negative pulses occur on the trailing edges. The differentiated signal is rectified to remove the positive pulses and generate the PPM waveform shown, which can be inverted to produce positive pulses. A simplified block diagram of a PPM system is shown in Figure 12–35.



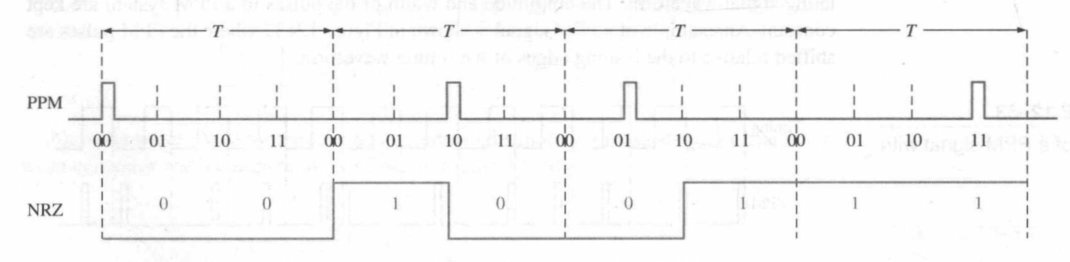
◄ **FIGURE 12–34**
A method of generating PPM.

▲ **FIGURE 12–35**
PPM system block diagram.

PPM is widely used in optical communications, such as fiber optics and in certain types of rf systems, such as radio control for model planes, boats, and cars. It is less sensitive to channel interference than PAM or PWM because noise can alter pulse amplitude and width but not so much the position.

PPM Encoding

A certain number of data bits (D) are encoded by a single pulse in one of 2^D possible positions during a specified fixed time period (T). The data rate is D/T bits per second (bps). Figure 12–36 illustrates the case of four time periods and two data bits per time period. There are $2^D = 2^2 = 4$ possible positions in each time period. As you can see, each position represents a 2-bit binary number. In the first time period, the pulse position represents 00, in the second time period the pulse position represents 10, et cetera. Any pulse could be in any of the four positions within each period, depending on the data being encoded. The code for this particular set of pulse positions is 00100111; it is shown in NRZ format in the figure.



▲ **FIGURE 12–36**
Encoding of a PPM signal.

EXAMPLE 12-6

For a PPM system with four data bits and a time period of 1 μs, determine the data rate. How many possible pulse positions are there in each time period?

Solution

The data rate is

$$\frac{D}{T} = \frac{4}{1\ \mu s} = \mathbf{4\ Mbps}$$

The number of possible pulse positions in each period is

$$2^D = 2^4 = \mathbf{16}$$

Related Problem

For eight data bits, what is the data rate ($T = 1\ \mu s$), and what is the number of possible pulse positions in each period?

Pulse Code Modulation

Pulse code modulation **(PCM)** involves sampling of an analog signal amplitude at regular intervals and converting the sampled values to a digital code. (Sampling was mentioned in Chapter 1, and the sample-and-hold process was covered in Chapter 11.) The concept of PCM is demonstrated in Figure 12–37.

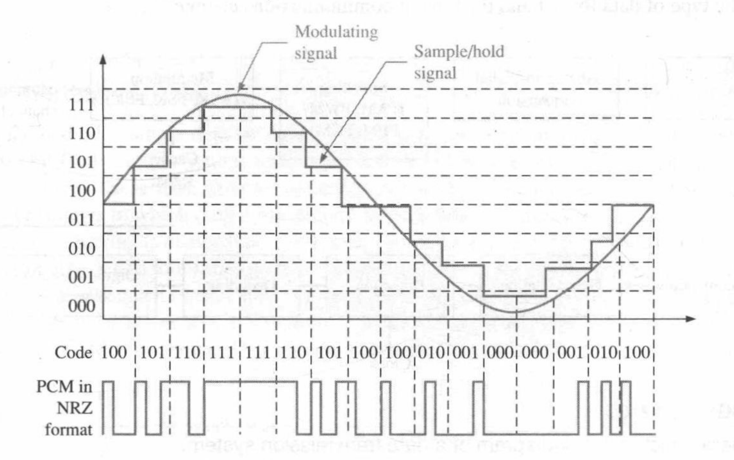


FIGURE 12–37
Concept of PCM with eight levels.

The modulating signal is a sine wave in this illustration, and its amplitude is divided into eight levels as shown. Each level is represented by a 3-bit binary number ($2^3 = 8$). The sine wave is sampled at fixed intervals; and the sampled value is held until the next sample, resulting in the green stair-step waveform that approximates the sine wave. The value at each sample is converted to a 3-bit binary number, and a pulse sequence is generated where each pulse represents a 1 and the absence of a pulse represents a 0. In this case, the PCM waveform is shown in NRZ format. The higher the sampling rate and the more levels used, the more accurate is the PCM representation.

PCM is used for digital audio in computers, Blu-ray, CD, and DVD formats. Also, it is used in digital telephone systems. A simplified block diagram of the PCM process is shown in Figure 12–38.

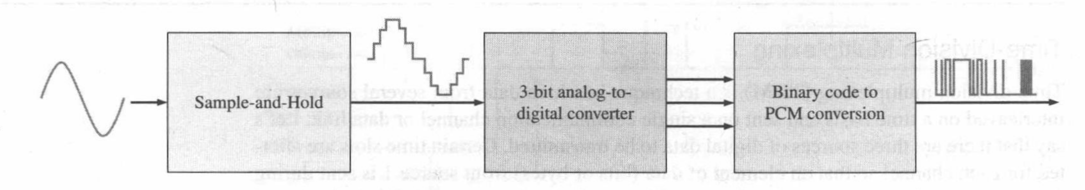


FIGURE 12–38
Block diagram of a PCM system.

EXAMPLE 12-7

In a PCM system with 32 levels, determine the number of code bits for each sample of an analog signal.

Solution

$$\text{Code bits} = 32 = 2^5$$

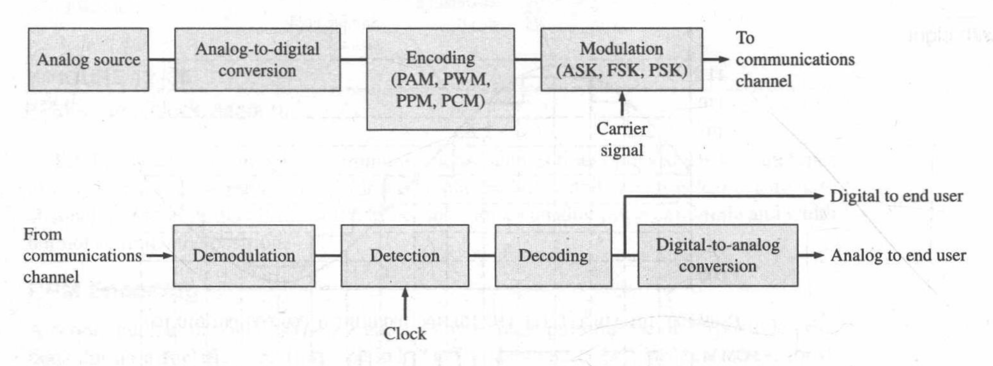
Five bits represent each sample.

Related Problem

What is the number of PCM code bits for each sample if the system has 64 levels?

Digital Data Systems

All digital data systems have certain common components and variations that depend on the type of data format. The three main data transmission combinations are digital-to-analog, analog-to-digital, and digital-to-digital. Figure 12–39 shows a general functional block diagram of a data transmission system. Each block is not used in all cases, depending on the type of data format and the type of communications channel.



▲ **FIGURE 12–39**

General function block diagram of a data transmission system.

SECTION 12-4 CHECKUP

1. List four types of pulse modulation methods.
2. What parameter is used in PAM to represent the value of the modulating signal?
3. What parameter is used in PWM to represent the value of the modulating signal?
4. What parameter is used in PPM to represent the value of the modulating signal?
5. What is used in PCM to represent the value of the modulating signal?

12-5 Multiplexing and Demultiplexing

Time-Division Multiplexing

Time-division multiplexing **(TDM)** is a technique in which data from several sources are interleaved on a time basis and sent on a single communication channel or data link. Let's say that there are three sources of digital data to be transmitted. Certain time slots are allotted for each channel so that an element of data (bits or bytes) from source 1 is sent during time slot 1, an element of data from source 2 is sent during time slot 2, and an element of data from source 3 is sent during time slot 3. This is repeated for time slots that follow until all the data have been sent. Figure 12–40 illustrates the TDM concept.

FIGURE 12–40
Basic concept of TDM.

Source 1 data | Source 2 data | Source 3 data | Source 1 data | Source 2 data
Time slot 1 | Time slot 2 | Time slot 3 | Time slot 4 | Time slot 5
Time

A simplified illustration of TDM is shown in Figure 12–41. Multiple data sources are switched (multiplexed) in a time sequence (t_1 through t_4) onto a single line (communications channel), and the single stream of data is switched back onto multiple lines in a synchronized time sequence. That is, data from source 1 go to the data 1 output during time slot t_1, data from source 2 go to the data 2 output during time slot t_2, and so on.

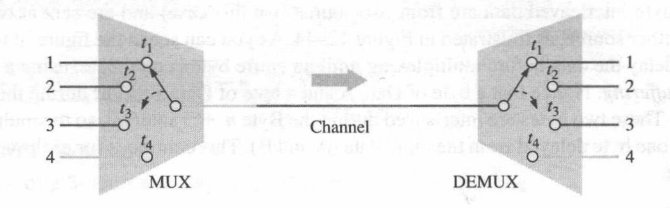


FIGURE 12–41
Simple illustration of TDM.

Bit-Interleaved TDM

In this method, a single data bit from a source is transmitted on the channel, followed by a data bit from another source, and so on. A time slot is reserved on the channel for each input source. These time slots are synchronized with the sender and receiver so that the receiver knows to which output the data bit in each time slot should go.

Bit-interleaving is demonstrated in Figure 12–42. In this case, the TDM channel data are transmitted at a rate four times greater than the data rate of the individual sources. Samples are sequentially taken of each data source (four in this case) during a bit time slot to determine if the bit is a 1 or a 0. The resulting values are sequentially placed onto the channel in 1, 2, 3, 4 order, as shown. This process is repeated for each of the bit times that follow.

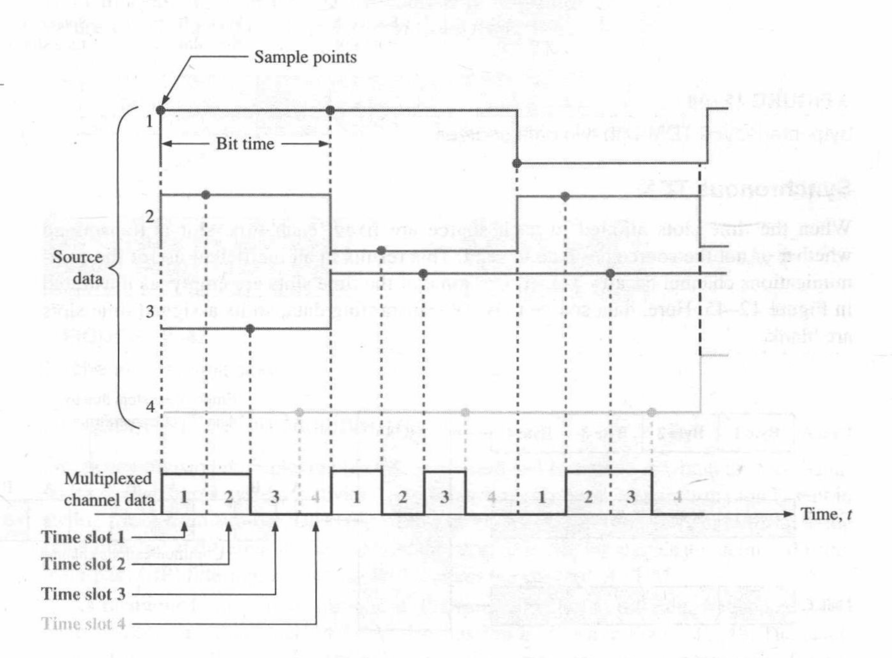
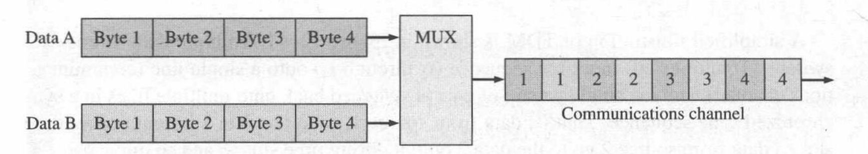


FIGURE 12–42
Illustration of TDM bit interleaving.

Byte-Interleaved TDM

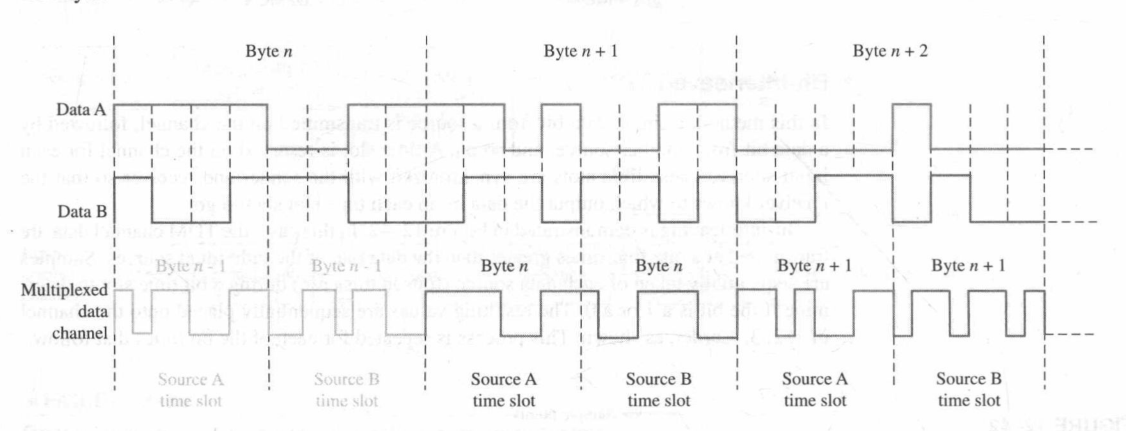
In this method, bytes for each input source are sequentially placed onto the data channel. As in bit-interleaving, synchronization between the mux and demux at each end of the communications channel is required. The basic concept is shown in Figure 12–43.



◀ **FIGURE 12–43**

Basic idea of byte-interleaved TDM.

The byte-interleaved data are from two sources (in this case) and are sent at twice the rate as either source, as illustrated in Figure 12–44. As you can see in the figure, it is necessary to delay the data before multiplexing until an entire byte is complete, using a process called *buffering*. Notice that a byte of Data A and a byte of Data B occur during the Byte n interval. These two bytes are interleaved during the Byte $n + 1$ interval, so the multiplexed data are one byte delayed from the input data (A and B). This continues for each successive data byte.

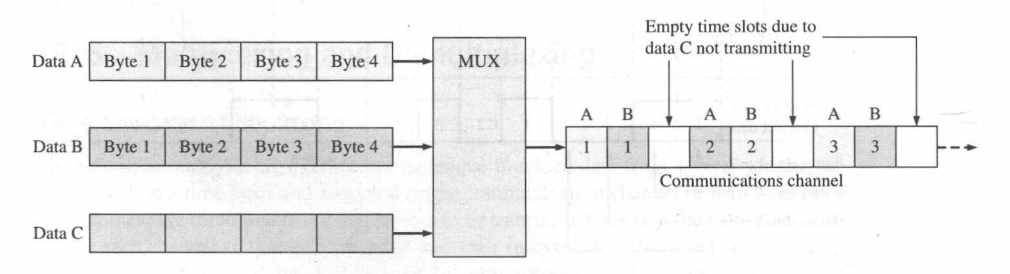


▲ **FIGURE 12–44**

Byte-interleaved TDM with two data sources.

Synchronous TDM

When the time slots allotted to each source are fixed, each time slot is transmitted whether or not the source has data to send. This results in an inefficient use of the communications channel because sometimes some of the time slots are empty, as illustrated in Figure 12–45. Here, data source C is not transmitting data, so its assigned time slots are blank.

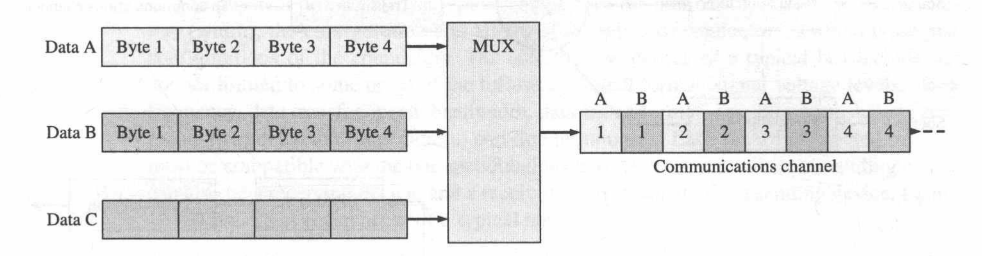


▲ **FIGURE 12–45**

Example of a 3-source synchronous TDM with one data source inactive.

Statistical TDM

The statistical TDM approach improves channel efficiency by making use of all the time slots. Only data from active sources are transmitted, so there are no blank time slots for inactive sources. The time slot assignment is variable rather than fixed, as in synchronous TDM. This method is shown in Figure 12–46 for the case where data source C is not transmitting. If data source C becomes active, the time slots are reassigned to accommodate the data.



▲ **FIGURE 12–46**

Example of a 3-source statistical TDM with one source inactive.

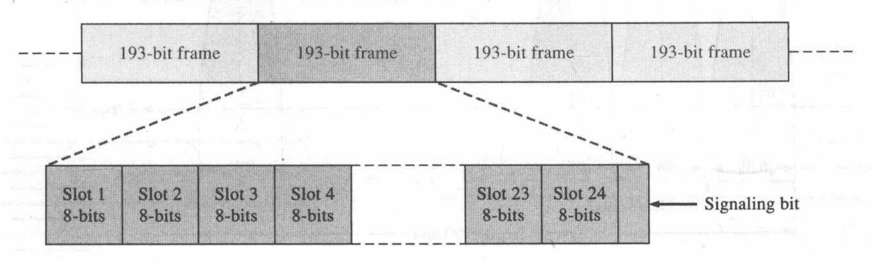
TDM is used by the telephone company in North America for nearly all voice traffic with what is known as the T1 system. A T1 line can carry 24 digitized telephone conversations and is capable of transmitting data at a rate of 1.544 Mbps. A voice signal is sampled 8,000 times per second, and each sample is converted to a byte of digital data. A voice signal requires a transmission rate of

$$\text{Voice transmission rate} = (8000\ \text{samples/s})(8\ \text{bits/sample}) = 64\ \text{kbps}$$

The number of digitized voice signals that can be multiplexed on a T1 line is

$$\text{Voice signals} = \frac{1.544\ \text{Mbps}}{64\ \text{kbps}} = 24$$

A T1 transmission over the channel consists of sequential 193-bit frames, as shown in Figure 12–47. Each frame is made up of twenty-four 8-bit slots plus one signaling bit.



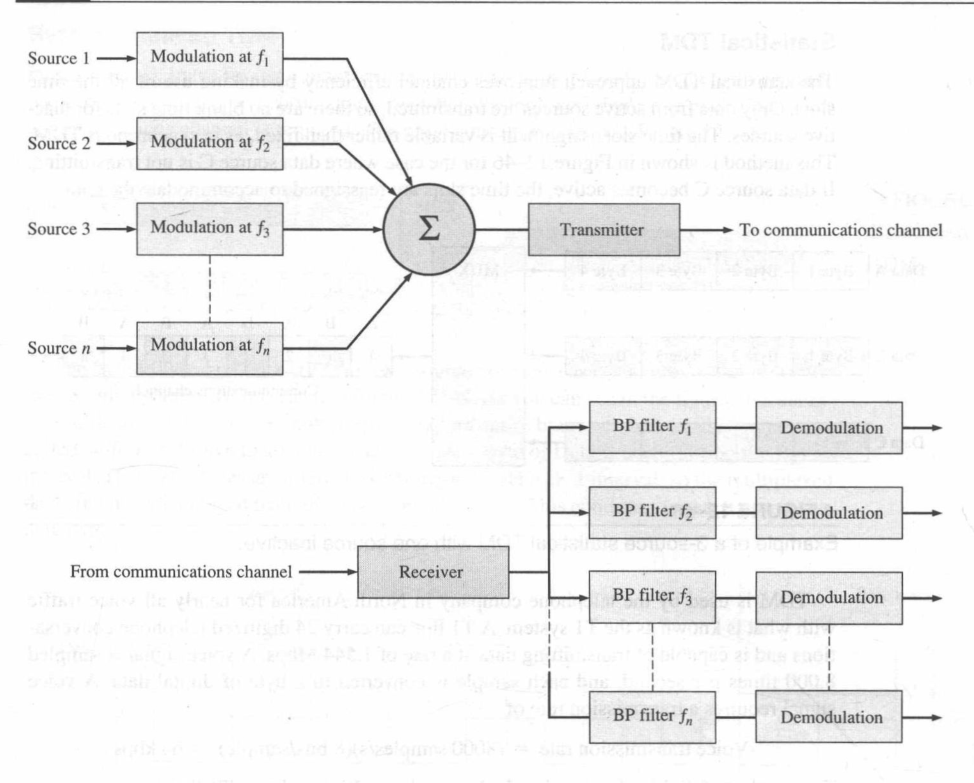
▲ **FIGURE 12–47**

T1 channel transmission.

Frequency-Division Multiplexing

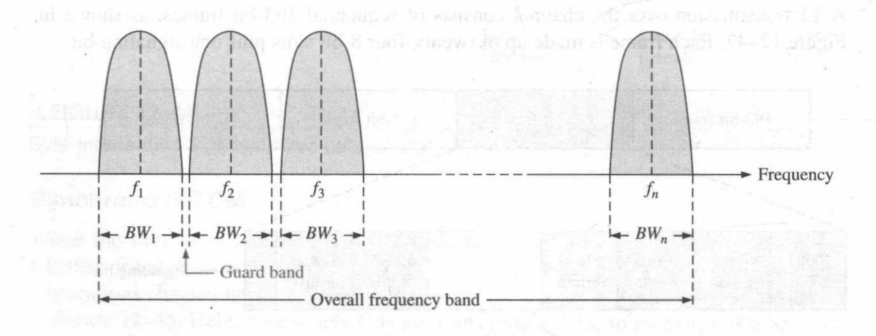
Frequency-division multiplexing **(FDM)** is a broadband technique in which the total bandwidth available to a system is divided into frequency sub-bands and information is sent in analog form. Each sub-band is assigned to a given source. The sources can transmit at the same time but at different frequencies. At the receiving end, the signals are demuxed using band-pass (BP) filtering. Figure 12–48 illustrates the concept of FDM.

As mentioned, all sources transmit at the same time but at different frequencies. The general spectrum of a composite FDM transmission is shown in Figure 12–49. The bandwidth (*BW*) of each source is centered around the carrier frequency for that source and is separated from the adjacent bandwidths by the guard band.



▲**FIGURE 12–48**

Basic FDM system. ∑ stands for summation.



▲**FIGURE 12–49**

Frequency spectrum of an FDM transmission.

SECTION 12-5 CHECKUP

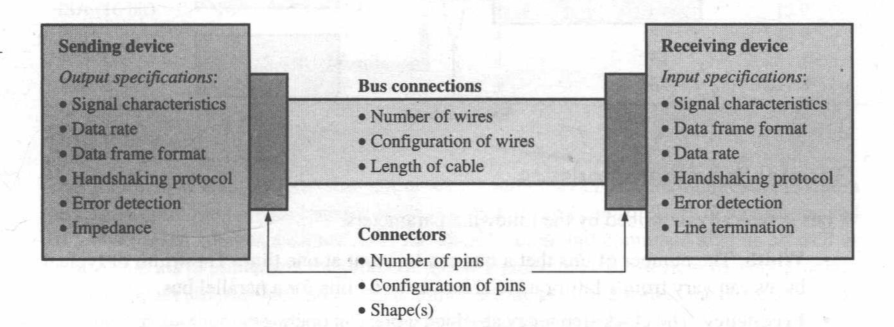
1. Discuss the reason that multiplexing is used in data communications.
2. What is TDM?
3. What is FDM?
4. Which has the higher efficiency, synchronous TDM or statistical TDM?
5. What is a guard band?

12-6 Bus Basics

The Bus

A bus allows two or more devices to communicate with each other, generally for the purpose of transmitting data. A bus is a set of physical wires and connectors and a set of electrical specifications. A bus can be either internal or external to a system.

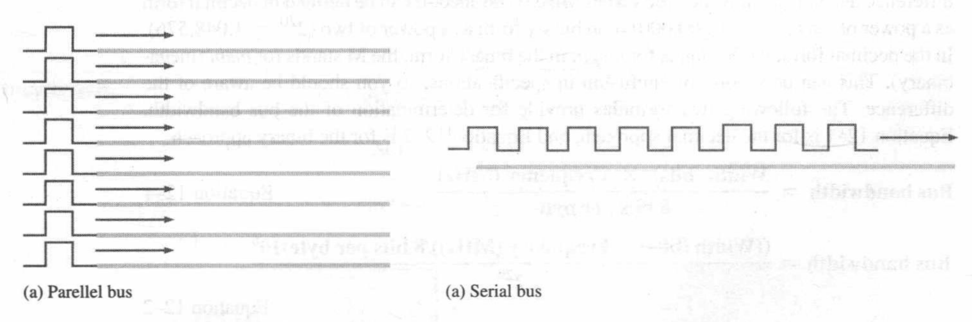
The physical properties of a typical bus include the number of wires or PCB conductors (width), the configuration and length of the wires or conductors, and the types and configurations of the connectors. The electrical properties of a typical bus include but are not limited to some or all of the following: signal format, signal voltage levels, clock frequency, data transfer speed, bandwidth, data frame format, data rate, handshaking protocol, error detection, impedances, and line termination. Each device connected to a bus must be compatible with the bus specifications in order to communicate. A sending device can also be a receiving device, and a receiving device can also be a sending device. Figure 12–50 illustrates the concept of a typical bus.



▶ FIGURE 12–50
Physical and electrical definition of a typical bus.

Parallel and Serial Buses

Buses can be either parallel or serial. A **parallel bus** carries data bits simultaneously, and a **serial bus** carries data bits sequentially one at a time. Figure 12–51 is a simple comparison of parallel and serial buses showing eight bits being transmitted.



▲ FIGURE 12–51
Comparison of parallel and serial buses.

It would seem that a parallel bus would transmit data faster than a serial bus because multiple data bits can be sent simultaneously. However, this is not always the case. As data rates increase, things like crosstalk across parallel bus lines, timing skew between bus lines, and EMI (electromagnetic interference) become problems that limit the speed. Serial buses are not limited by those factors and can actually transmit data at higher rates than parallel buses in many situations.

Internal and External Buses

Internal buses carry information within a system, that is, from one part of the system to another part of the same system. External buses (also known as *expansion buses*) are used to connect one system to another separate system. For example, a computer connects to peripheral units such as a monitor, keyboard, mouse, and printer through external buses, as illustrated in Figure 12–52.

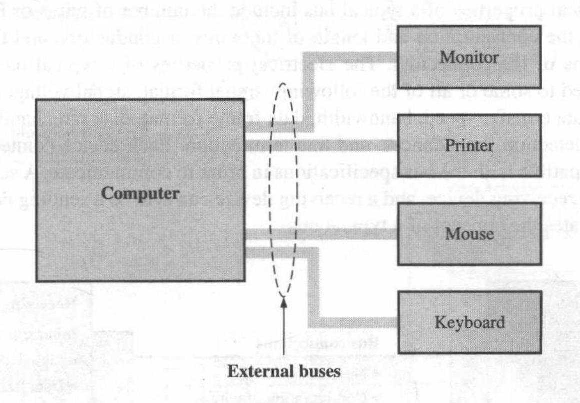


FIGURE 12–52
Example of external bus application.

General Bus Characteristics

A bus is typically described by the following parameters:

- **Width** The number of bits that a bus can transmit at one time. The width of typical buses can vary from 1 bit for a serial bus up to 64 bits for a parallel bus.
- **Frequency** The clock frequency at which a bus can operate
- **Transfer speed** The number of bytes per clock cycle
- **Bandwidth** The number of bytes per clock cycle times the number of clock cycles per second; that is, transfer speed times frequency. Bus bandwidth is sometimes called *throughput*.

Bus bandwidth can be specified in two ways, which result in slightly different values. The difference depends on how the prefix M in MBps is defined. It can be defined in decimal form as a power of ten ($10^6 = 1{,}000{,}000$) or in binary form as a power of two ($2^{20} = 1{,}048{,}576$). In the decimal form, the M stands for *mega*; in the binary form, the M stands for *mebi* (mega-binary). This can be a point of confusion in specifications, so you should be aware of the difference. The following two formulas provide for determination of the bus bandwidth. Equation 12–1 is for the decimal approach, and Equation 12–2 is for the binary approach.

$$\textbf{Bus bandwidth} = \frac{\textbf{Width (bits)} \times \textbf{Frequency (MHz)}}{\textbf{8 bits per byte}} \qquad \text{Equation 12–1}$$

$$\textbf{Bus bandwidth} = \frac{((\textbf{Width (bits)} \times \textbf{Frequency (MHz)})/\textbf{8 bits per byte})\mathbf{10^6}}{\mathbf{2^{20}}}$$

Equation 12–2

EXAMPLE 12-8

A certain bus is specified with a width of 32 bits and a frequency of 66 MHz. Determine the bus bandwidth expressed as two different values, according to the decimal and binary definitions of M. Note that Bps is bytes per second.

Solution

Using the decimal definition of M (10^6) in the unit of MBps,

$$\text{Bandwidth} = \frac{32 \text{ bits} \times 66 \text{ MHz}}{8 \text{ bits per byte}} = \mathbf{264\ MBps}$$

Using the binary definition of M (2^{20}),

$$\text{Bandwidth} = \frac{((32 \text{ bits} \times 66 \text{ MHz})/8 \text{ bits per byte})\ 10^6}{2^{20}} = \mathbf{252\ MBps}$$

Related Problem

What is the frequency for a bus that has a width of 64 bits and bandwidth of 125 MBps as specified in decimal form?

Table 12–1 lists some typical buses and their characteristics.

TABLE 12–1

Bus	Width (bits)	Frequency (MHz)	Transfer Speed (bytes/cycle)	Bandwidth (MBps) Decimal	Bandwidth (MBps) Binary
ISA (16 bit)	16	8.3	2	16.6	15.9
PCI	32	33	4	132	125.9
PCI-X	32	66	4	264	251.8
AGP	32	66	4	264	251.8

Bus Protocol

Bus protocol is a set of rules that allow two or more devices to communicate through a bus. Buses provide for data transfer, address selection, and control. Each device connected to a bus has an address assigned to it for identification and command signals as well as control signals to implement the protocol. These signals allow the devices to work properly together by identifying each other and communicating back and forth. One device can send a request to another device and get an acknowledgement or reply.

Handshaking

The **handshake** is a routine by which two devices initiate and complete a bus transfer. Figure 12–53 shows a simple handshake process, including a timing diagram, in which a requesting device (sometimes called the *master*) and a responding device (sometimes called the *servant*) initiate and complete a data transfer.

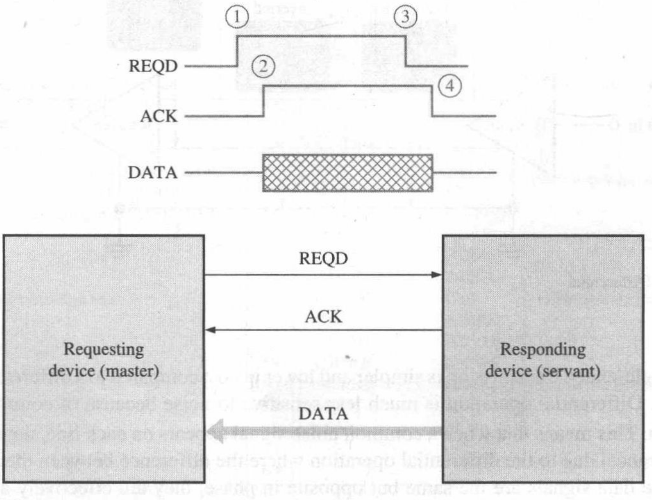


FIGURE 12–53
Simple example of handshake and data transfer.

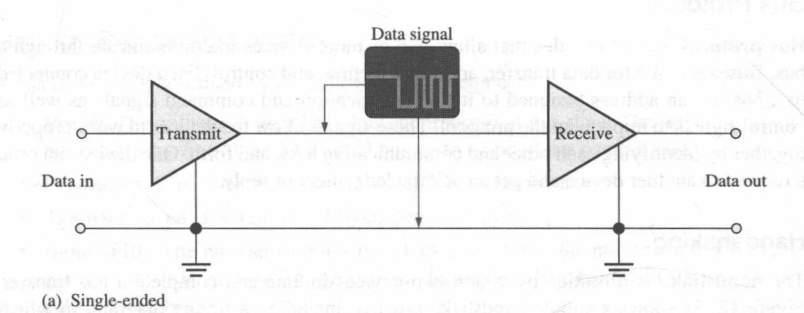
Synchronous and Asynchronous Buses

A synchronized bus includes a clock in the control lines and has a fixed protocol that is relative to the clock. A synchronous bus is fast but has the disadvantage that every device connected to it must operate at the same clock frequency. Also, the physical length of the bus may be limited because of having to carry a high-frequency clock signal.

An asynchronous bus is not clocked, so it can serve various devices with different clock rates. The asynchronous bus uses a handshake protocol to establish communication, as previously described.

Single-Ended vs. Differential Buses

Data communications between devices can be classified as either single-ended or differential in terms of the physical bus configuration. In general, single-ended operation is limited in both data rate and distance (cable length). Differential operation provides much higher data rates and longer transmission distances. **Single-ended operation** uses one wire for data and one wire for ground, where the signal voltage on the wire is with respect to ground. **Differential operation** uses two wires for data and one wire for ground. The data signal is sent on one wire in the twisted pair and its complement (inversion) is sent on the other wire. The difference between the two data wires is the differential signal. Figure 12–54 shows both single-ended and differential operation.



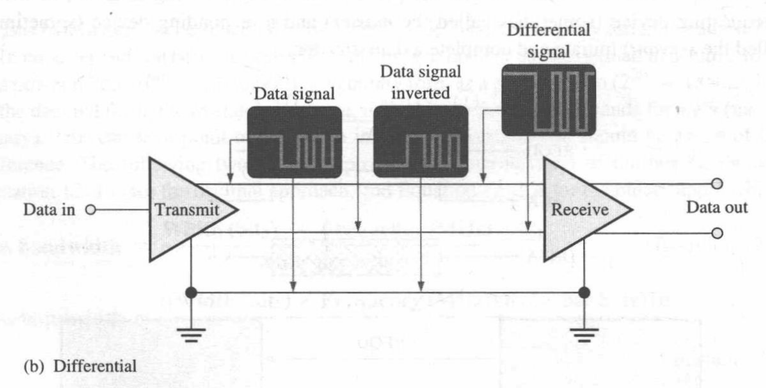


FIGURE 12–54 A comparison of single-ended and differential bus operation.

A single-ended transmission is simpler and lower in cost compared to a differential transmission. Differential operation is much less sensitive to noise because of common-mode rejection. This means that when a common noise signal appears on each line, the two noise signals cancel due to the differential operation where the difference between them is zero. Since the data signals are the same but opposite in phase, they are effectively added and the data signal is reinforced while the noise signal is cancelled. Due to the twisted pair, crosstalk is reduced at higher frequencies, thus allowing longer cables.

SECTION 12-6 CHECKUP

1. List two factors that may limit the speed of a parallel bus.
2. What is the basic difference between an internal and an external bus?
3. Name four bus characteristics.
4. What is bus protocol?
5. Discuss the difference between single-ended and differential bus operation.

12-7 Parallel Buses

The PCI Bus

The **PCI** (peripheral component interconnect) **bus** is an internal synchronous bus for interconnecting chips, expansion boards, and processor/memory subsystems. The original PCI bus had a width of 32 bits and a frequency of 33 MHz. Another version has a width of 64 bits and a frequency of 66 MHz. Still later versions enable 64-bit data transfers using up to a 133 MHz clock to enable bandwidths of up to 1066 MBps.

The original PCI standard required 5 V power and signal levels. As the standard evolved, the option for 3.3 V was added. The latest standard provides for 3.3 V only. The 32-bit PCI connector has 62 pins and 124 contacts (62 per side). Thirty-two of the contacts are used for both a 32-address and 32 bits of data, which are multiplexed. The remaining pins are used for command and control signals, power, ground, etc. A 64-bit PCI connector has an additional 32 pins for a total of 94. The 32-bit PCI connector is shown in Figure 12–55. It has 64 pins of which 62 are used.

PIN	SIDE A	SIDE B	
1	−12 V	TRST#	
2	TCK	+12 V	JTAG port pins
3	Gnd	TMD	
4	TDO	TDI	
5	+5 V	+5 V	
6	+5 V	INTA#	
7	INTB#	INTC#	Interrupt lines
8	INTD#	+5 V	
9	PRSNT1#	Reserved	Indicates 7.5 or 24 W power
10	Reserved	IOPWR	+5 V or +3.3 V
11	PRSNT1#	Reserved	Indicates 7.5 or 24 W power
12	Gnd	Gnd	
13	Gnd	Gnd	Key notch for 3.3 V cards
14	Reserved	3.3 Vaux	Standby power
15	Gnd	RST#	Bus reset
16	CLK	IOPWR	33 MHz/66 MHz clock
17	Gnd	GNT#	Bus grant motherboard to card
18	REQ#	Gnd	Bus request card to motherboard
19	IOPWR	PME#	Power management event/3.3 V
20	AD[31]	AD[30]	
21	AD[29]	+3.3 V	
22	Gnd	AD[28]	
23	AD[27]	AD[26]	Address/data bus
24	AD[25]	Gnd	
25	+3.3 V	AD[24]	
26	C/BE[3]#	IDSEL	
27	AD[23]	+3.3 V	
28	Gnd	AD[22]	
29	AD[21]	AD[20]	
30	AD[19]	Gnd	
31	+3.3 V	AD[18]	
32	AD[17]	AD[16]	

PIN	SIDE A	SIDE B	
33	C/BE[2]#	+3.3 V	
34	Gnd	FRAME#	Bus transfer in progress
35	IRDY#	Gnd	Initiator ready
36	+3.3 V	TRDY#	Target ready
37	DEVSEL#	Gnd	Target selected
38	Gnd	STOP#	Target halt request
39	LOCK#	+3.3 V	Locked transaction
40	PERR#	SMBCLK	Parity error/SMBus clock
41	+3.3 V	SMBDAT	SMBus data
42	SERR#	Gnd	System error
43	+3.3 V	PAR	Even parity over AD bus
44	C/BE[1]#	AD[15]	
45	AD[14]	+3.3 V	
46	Gnd	AD[13]	Address/data bus
47	AD[12]	AD[11]	
48	AD[10]	Gnd	
49	MGGEN/Gen	AD[09]	
50	Gnd	Gnd	Key notch for +5 V cards
51	Gnd	Gnd	
52	AD[08]	C/BE[0]#	
53	AD[07]	+3.3 V	
54	+3.3 V	AD[06]	
55	AD[05]	AD[04]	
56	AD[03]	Gnd	Address/data bus
57	Gnd	AD[02]	
58	AD[01]	AD[00]	
59	IOPWR	IOPWR	
60	ACK64#	REQ64#	For 64-bit expansion
61	+5 V	+5 V	
62	+5 V	+5 V	
63			
64			

▲ **FIGURE 12–55**

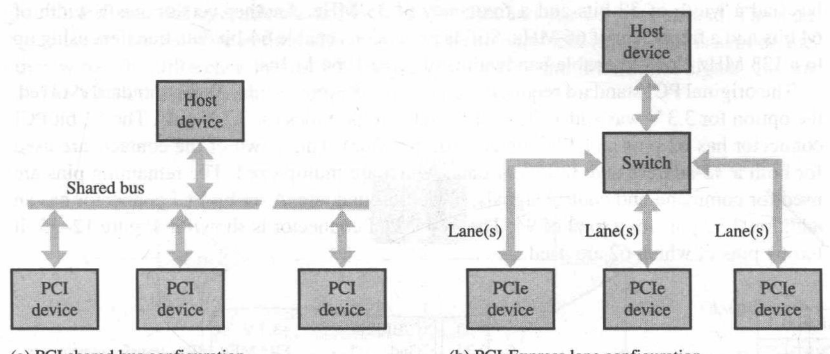
Pin layout and functional designation for a 32-bit PCI connector.

The PCI-X Bus

The **PCI-X bus** is a high-performance enhancement of the PCI and is backward compatible with the PCI bus, although it is a faster bus and has some additional features. A 64-bit bus, the PCI-X runs at a frequency of 133 MHz. The PCI-X 2.0 revision supports frequencies of 266 MHz and 533 MHz. Some additional features increase system reliability by minimizing errors at high transfer rates. Servers are the major application for the PCI-X.

The PCI-Express Bus

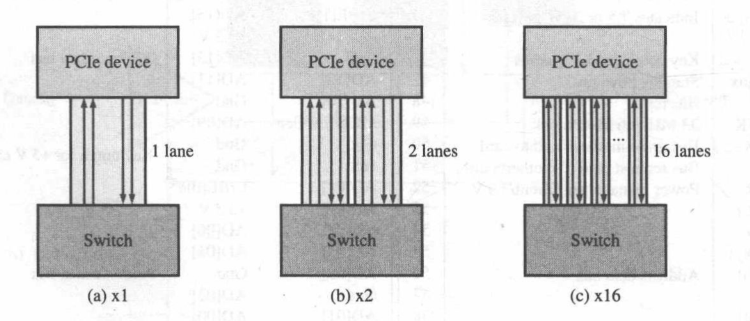
The **PCI-Express** is also designated as PCIe or PCI-E. This bus differs from the PCI and PCI-X buses in that it does not use a **shared bus**. Both PCI and PCI-X use a shared bus configuration, as shown in Figure 12–56(a). Each PCIe device has a dedicated path, called a *lane*, to a single chip known as a *switch*, as shown in part (b). More lanes result in a faster data transfer. High speed makes PCI-Express ideal for video and graphics applications.



(a) PCI shared bus configuration (b) PCI-Express lane configuration

◀ **FIGURE 12–56**
Comparison of PCI and PCI-Express.

A single PCI-Express lane contains two pairs of conductors. One pair of conductors from a given device receives data and the other pair sends data in a serial format. A single lane configuration is known as x1 and is illustrated in Figure 12–57(a). A x2 configuration is shown in part (b), and a x16 configuration is shown in part (c). Other possible configurations are x4, x8, and x32.



(a) x1 (b) x2 (c) x16

◀ **FIGURE 12–57**
PCI-Express lane configurations.

Bidirectional data are transferred serially on each lane. For multiple-lane configurations, the serial data on each lane are in parallel with the serial data on all the other lanes. The data are transferred at one bit per cycle on a x1 connection, two bits per cycle on a x2, and sixteen bits per cycle on a x16. The PCI-Express has a bandwidth of 4 Gbps per lane. PCI-Express supports hot swapping in which expansion cards can be added or removed without turning off the system. PCI and PCIe are software compatible but not hardware compatible.

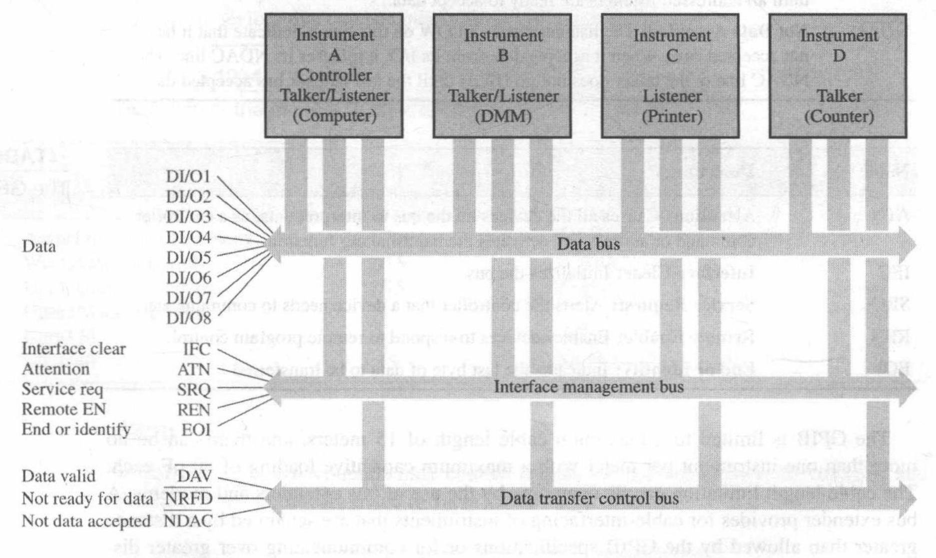
The IEEE-488 Bus

This bus standard has been around a long time and is also known as the General-Purpose Interface Bus (**GPIB**). Widely used in test and measurement applications, it was developed by Hewlett-Packard in the 1960s. The IEEE 488 specifies 24 lines that are used to transfer eight parallel data bits at a time and provide eight control signals that include three handshake lines and five bus-management lines. Also included are eight ground lines used for shielding and ground returns. The maximum data transfer rate for the IEEE 488 standard is 1 Mbps. A superset of this standard, called the HS488, has a maximum data rate of 8 Mbps.

To connect test equipment to a computer using the IEEE-488 bus, an interface card is installed in the computer, which turns the computer into a system controller. In a typical GPIB setup, up to 14 controlled devices (test and measurement instruments) can be connected to the system controller. When the system controller issues a command for a controlled device to perform a specified operation, such as a frequency measurement, it is said that the controller "talks" and the controlled device "listens."

A **listener** is an instrument capable of receiving data over the GPIB when it is addressed by the system controller (computer). Examples of listeners are printers, monitors, programmable power supplies, and programmable signal generators. A **talker** is an instrument capable of sending data over the GPIB. Examples are DMMs and frequency counters that can output bus-compatible data. Some instruments can send and receive data and are called talker/listeners; examples are computers, modems, and certain measurement instruments. The system controller can specify each of the other instruments on the bus as either a talker or a listener for the purpose of data transfer. The controller is usually a talker/listener.

A typical GPIB arrangement is shown in Figure 12–58 as an example. The three basic bus signal groupings are shown as the *data bus, data transfer control bus,* and *interface management bus.*



▲**FIGURE 12–58**

A typical IEEE 488 (GPIB) connection.

The parallel data lines are designated DI/O1 through DI/O8 (data input/output). One byte of data is transferred on this bidirectional part of the bus. Every byte that is transferred undergoes a handshaking operation via the data transfer control. The three active-LOW handshaking lines indicate if data are valid ($\overline{\text{DAV}}$), if the addressed instrument is not ready for data ($\overline{\text{NRFD}}$), or if the data are not accepted ($\overline{\text{NDAC}}$). More than one instrument can accept data at the same time, and the slowest instrument sets the rate of transfer. Figure 12–59 shows the timing diagram for the GPIB handshaking sequence, and Table 12–2 describes the handshaking signals.

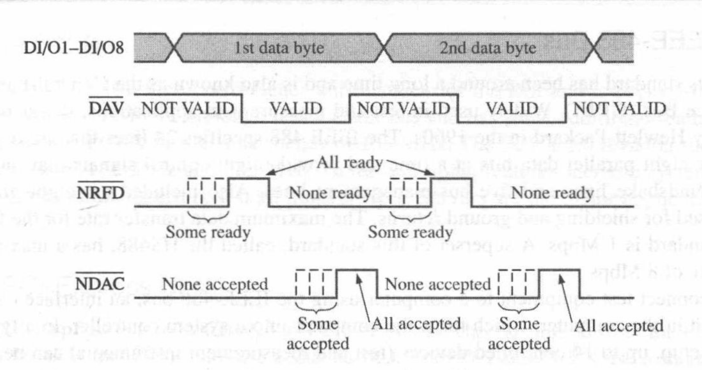


FIGURE 12–59
Timing diagram for the GPIB handshaking sequence.

The five signals of the interface management bus control the orderly flow of data. The ATN (attention) line is monitored by all instruments on the bus. When ATN is active, the system controller selects the specific interface operation, designates the talkers and the listeners, and provides specific addressing for the listeners. Each GPIB instrument has a specific identifying address that is used by the system controller. Table 12–3 describes the GPIB interface management lines and their functions.

TABLE 12–2
The GPIB handshaking signals.

Name	Description
$\overline{DAV}$	**Data Valid:** After the talker detects a HIGH on the $\overline{NRFD}$ line, a LOW is placed on this line by the talker when the data on its I/O are settled and valid.
$\overline{NRFD}$	**Not Ready for Data:** The listener places a LOW on this line to indicate that it is not ready for data. A HIGH indicates that it is ready. The $\overline{NRFD}$ line will not go HIGH until all addressed listeners are ready to accept data.
$\overline{NDAC}$	**Not Data Accepted:** The listener places a LOW on this line to indicate that it has not accepted data. When it accepts data from its I/O, it releases its $\overline{NDAC}$ line. The $\overline{NDAC}$ line to the talker does not go HIGH until the last listener has accepted data.

TABLE 12–3
The GPIB management lines.

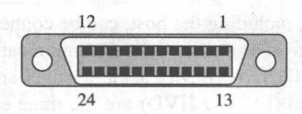
Name	Description
ATN	**Attention:** Causes all the devices on the bus to interpret data, as a controller command or address and activates the handshaking function.
IFC	**Interface Clear:** Initializes the bus.
SRQ	**Service Request:** Alerts the controller that a device needs to communicate.
REN	**Remote Enable:** Enables devices to respond to remote program control.
EOI	**End or Identify:** Indicates the last byte of data to be transferred.

The GPIB is limited to a maximum cable length of 15 meters, and there can be no more than one instrument per meter with a maximum capacitive loading of 50 pF each. The cable length limitation can be overcome by the use of bus extenders and modems. A bus extender provides for cable-interfacing of instruments that are separated by a distance greater than allowed by the GPIB specifications or for communicating over greater distances via modem-interfaced telephone lines.

The IEEE-488 connector and pin configuration are shown in Figure 12–60.

The Parallel SCSI Bus

The **SCSI** (small computer system interface) bus, generally pronounced "skuzy", is a parallel I/O bus with a width of either 8, 16, or 32 bits, depending on the version. For many years SCSI has been one of the most widely used buses in storage servers and data centers. SCSI is also used for the purpose of transferring data between a computer and peripheral devices, such as hard disks, tape drives, scanners, and CD drives. Figure 12–61 shows the SCSI symbol.



Pin	Designation	Description	Pin	Designation	Description
1	DIO1	Data input/output bit.	13	DIO5	Data input/output bit.
2	DIO2	Data input/output bit.	14	DIO6	Data input/output bit.
3	DIO3	Data input/output bit.	15	DIO7	Data input/output bit.
4	DIO4	Data input/output bit.	16	DIO8	Data input/output bit.
5	EOI	End-or-identify.	17	REN	Remote enable.
6	DAV	Data valid.	18	GND	(wire twisted with DAV)
7	NRFD	Not ready for data.	19	GND	(wire twisted with NRFD)
8	NDAC	Not data accepted.	20	GND	(wire twisted with NDAC)
9	IFC	Interface clear.	21	GND	(wire twisted with IFC)
10	SRQ	Service request.	22	GND	(wire twisted with SRQ)
11	ATN	Attention.	23	GND	(wire twisted with ATN)
12	SHIELD		24	Logic ground	

▲ **FIGURE 12–60**

The IEEE-488 (GPIB) bus connector and pin assignments.

▲ **FIGURE 12–61**

SCSI symbol.

The original version of the SCSI parallel bus standard was introduced in 1986 and designated SCSI-1. The current SCSI standard is known as SCSI-5, which was preceded by SCSI-1, SCSI-2, SCSI-3, and SCSI-4. Later versions are backward compatible with earlier versions. There are many variations of the SCSI standard version with designations such as *asynchronous, synchronous, fast, ultra,* and *wide,* which have different speeds, widths, and number of devices that can be connected, as shown in Table 12–4.

▼ **TABLE 12–4**

Evolution of the parallel SCSI standard.

Version	Variations	Maximum Target Devices Connected	Bus Width	Data Transfer Rates
SCSI-1	Asynchronous/Synchronous	7	8 bits	4 MBps/5 MBps
SCSI-2	Wide, Fast, Fast/Wide	7/15	8/16 bits	10 MBps/20 MBps
SCSI-3	Ultra, Ultra/Wide, Ultra2, Ultra2/Wide, Ultra160	7/15	8/16 bits	20 MBps/40 MBps/ 80 MBps/160 MBps
SCSI-4	Ultra320	15	16 bits	320 MBps
SCSI-5	Ultra640	15	8/16/32 bits	640 MBps

SCSI Signals

A parallel SCSI bus contains nine control signals in addition to data, dc voltages, and ground. These signals are listed in Table 12–5.

► **TABLE 12–5**

SCSI parallel bus signals.

Signal	Description
BSY	Busy, Bus in use
SEL	Select
RST	Reset
C/D	Control/Data
MSG	Message
REQ	Request
ACK	Acknowledge a request
ATN	Attention
I/O	Input or output

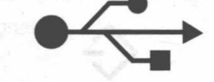
Up to eight or sixteen devices, including the host, can be connected to a SCSI bus, but only two devices can communicate at any given time. Communication begins when an initiating device sends a request and the target device acknowledges and performs the request. Single-ended (SE) and differential (LVD or HVD) are the three electrical specifications. Single-ended operation is limited to a cable length of 6 meters, and differential operation allows up to 25 meters. LVD is low-voltage differential and HVD is high-voltage differential. SCSI devices can operate either asynchronously or synchronously. The serial SCSI bus is introduced in Section 12-9.

SECTION 12-7 CHECKUP

1. What does PCI stand for?
2. List two alternate designations for the PCI-Express bus.
3. What is a *lane*?
4. What is the IEEE designation for the GPIB?
5. What does SCSI stand for?

12-8 The Universal Serial Bus (USB)

The **universal serial bus (USB)** is a widely used standard serial bus for connecting peripherals to a computer. There are typically two or more USB ports on computers and, with USB hubs, up to 127 devices can be connected. USB allows the devices to be connected or disconnected while the computer is running (hot swapping). Figure 12–62 shows the symbol for USB.



▲FIGURE 12–62
USB symbol.

The original USB standard was 1.0, which was followed by 1.1. USB 2.0 replaced the two original versions and more recently USB 3.0 was introduced. The earlier versions are still in use, especially 2.0. In terms of data rate, USB has four classifications: low-speed, full-speed, high-speed, and super-speed. Table 12–6 shows how the data rate classifications apply to each of the USB versions and Table 12–7 shows the data rate values.

▼TABLE 12–6

	Low-Speed	Full-Speed	High-Speed	Super-Speed
USB 1.0	•			
USB 1.1	•	•		
USB 2.0	•	•	•	
USB 3.0	•	•	•	•

▼TABLE 12–7

Data Rate	Maximum Value
Low-speed	0.1875 MBps
Full-speed	1.5 MBps
High-speed	60 MBps
Super-speed	625 MBps

Cable length is an important specification for buses. Table 12–8 lists maximum cable lengths for three USB versions and maximum total lengths when multiple cables are strung together using USB hubs. A hub is a common connection device with multiple ports.

◄TABLE 12–8

	USB 1.1	USB 2.0	USB 3.0
Max cable length	9.8 ft. (3.0 m)	16.4 ft. (5.0 m)	9.8 ft. (3.0 m)
Maximum total length	49.2 ft. (15 m)	82.0 ft. (25 m)	49.2 ft. (15 m)

USB Cable and Connectors

USB versions up to and including 2.0 have a four-wire cable that includes a twisted pair to reduce or eliminate noise for data transmission, a +5 V wire, and a ground wire color-coded, as shown in Figure 12–63(a). The standard type A and type B connectors are shown in parts (b) and (c) with pin designations. USB hosts (computer) and devices (peripherals)

have sockets, and all USB cables have a type A plug at one end and a type B plug at the other. The sockets on a host are Type A, and the sockets on peripheral devices are Type B. Hubs have both Type A and Type B. The USB standard also specifies smaller connectors designated mini and micro.

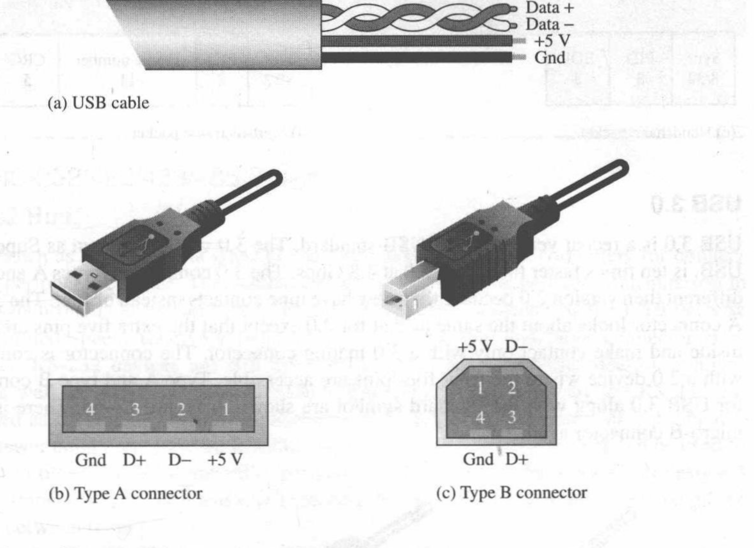


FIGURE 12–63
USB cable and connectors for USB standards through 2.0.

USB Data Format

Serial data are transmitted on the twisted pair (D+ and D−) using half-duplex differential mode to minimize EMI and improve the signal-to-noise ratio. Data are sent in packets using NRZI (non-return-to-zero invert) encoding format with a 3.3 V level (differentially, there are 6.6 V between the two data lines). A packet format can contain the following fields:

Sync field All packets start with a sync (synchronization) field. The sync field consists of 8 bits for low and 32 bits (full speed) for high speed and is used to synchronize the receiver clock with that of the transmitter.

PID field The packet identification field is used to identify the type of packet that is being transmitted. There are four bits in the PID; however, to ensure it is received correctly, the four bits are complemented and repeated, making an 8-bit PID code.

ADDR field The address field specifies to which device the packet is sent. The seven bits in this field allow for 127 devices to be supported. Address 0 is invalid.

Data field The data field contains up to 1024 bytes of data.

ENDP field The endpoint field is made up of four bits, allowing 16 possible endpoints. An endpoint is a data source or load. Low-speed devices, however, can only have two additional endpoints on top of the default pipe (four endpoints max). Endpoints can be described as sources or sinks of data.

CRC field Cyclic redundancy checks are performed on the data within the packet using from 5–16 bits, depending on the type of packet.

EOP field This packet field signals the end of a packet.

Four types of USB packets are token, data, handshake, and start-of-frame, as shown in Figure 12–64 with the packet format for each type. Each field is labeled and the number of bits shown. The token packet indicates the type of transaction, the data packet contains the actual data, the handshake packet acknowledges a transaction, and the start-of-frame packet begins a new frame. The token packet, data packet, handshake packet, and start-of-frame packet each have a different packet format as specified by the PID field.

Sync 8/32	PID 8	ADDR 7	ENDP 4	CRC 5	EOP 3

(a) Token packet

Sync 8/32	PID 8	Data 0-8192	CRC 16	EOP 3

(b) Data packet

Sync 8/32	PID 8	EOP 3

(c) Handshake packet

Sync 8/32	PID 8	Frame number 11	CRC 5	EOP 3

(d) Start-of-frame packet

FIGURE 12–64 Types of USB packets.

USB 3.0

USB 3.0 is a recent version of the USB standard. The 3.0 version, known as SuperSpeed USB, is ten times faster than USB 2.0 at 4.8 Gbps. The 3.0 connectors (types A and B) are different then version 2.0 because they now have nine contacts instead of four. The 3.0 type A connector looks about the same as that for 2.0 except that the extra five pins are further inside and make contact only with a 3.0 mating connector. The connector is compatible with a 2.0 device where the front four pins are accessible. Type A and type B connectors for USB 3.0 along with the standard symbol are shown in Figure 12–65. There is also a micro-B connector available.

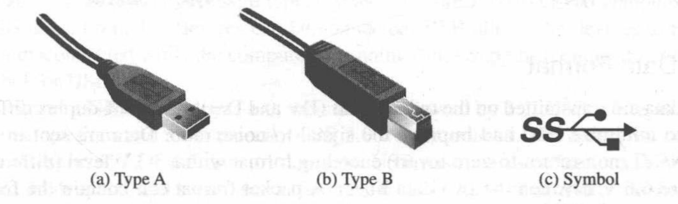


FIGURE 12–65 USB 3.0 connectors and symbol.

USB 3.0 is, for the most part, backward compatible with USB 2.0, but the speed is limited to the 2.0 specification. The USB 3.0 cable consists of two additional twisted pairs for data and an additional ground, for a total of nine wires. Unlike the previous versions, version 3.0 is full-duplex, meaning that data can be sent and received simultaneously. One twisted pair is for receiving data, and two additional twisted pairs are for sending high-speed data.

The recent USB 3.1 specification increases the data rate to 10 Gbps, twice that of the 3.0. USB 3.1 is backward compatible with 3.0 and 2.0 and a new connector, type C, is included in the specification.

Figure 12–66 shows the USB in a typical computer system. The computer acts as a host and uses Type A connectors. The hub functions as both a host and a device.

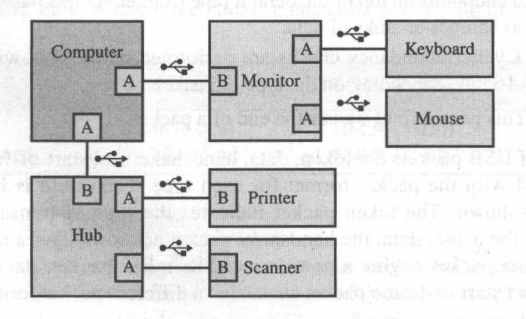


FIGURE 12–66 Example of USB applications.

SECTION 12-8 CHECKUP

1. What does USB stand for?
2. What are the functions of the four pins in a USB 2.0 connector?
3. Why are twisted pairs used in USB cables?
4. Describe the basic differences between USB 2.0 and USB 3.0.

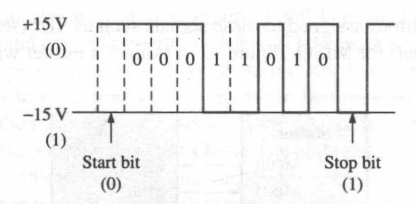
12-9 Other Serial Buses

The RS-232/422/423/485 Buses

RS-232 Bus

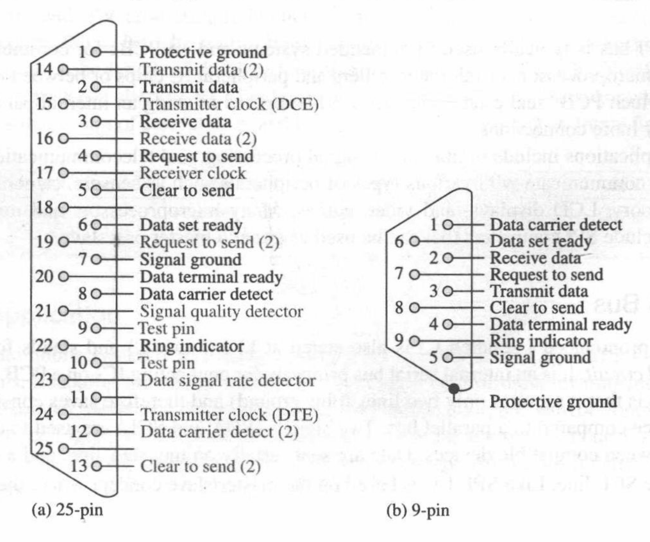
Also known as **EIA-232**, the **RS-232 bus** was once standard on computers for connection to peripheral devices. The standard provides for single-ended data transmission in either synchronous or asynchronous formats. It has been replaced by the USB because of its limited speed, relatively large voltage requirements, and large connector size. However, RS-232 devices are still used in industrial and telecommunication applications as well as scientific instrumentation. The devices connected by the RS-232 are classified as DTE (data terminal equipment) or DCE (data communication equipment). Since newer computers have no RS-232 ports, USB-to-RS-232 converters can be used to connect to older RS-232 compatible peripherals, if necessary. The standard is designed for one transmitting device and one receiving device with a maximum cable length of 50 feet between them.

The maximum RS-232 data rate is 20 kbps. The data format typically consists of seven or eight bits of data, a start bit, a parity bit in some cases depending on the protocol, and a stop bit. A transmitted signal level between +5 V and +15 V represents a binary 0 and between −5 V and −15 V represents a binary 1. The data is transmitted in NRZ format, as Figure 12–67 shows.



▶ FIGURE 12–67
Example of RS-232 transmitted data format. A parity bit is not included.

The standard 25-pin connector for RS-232 is shown in Figure 12–68(a). A smaller 9-pin connector, is shown in part (b).



▶ FIGURE 12–68
Standard RS-232 connectors.

RS-422/423/485

The **RS-422 bus** provides for differential transmission for greater distances (longer cable length) and has higher data rates than the RS-232 bus. Also, the standard defines the number of receiving devices as ten for a line with one driver (transmitting device) compared to one receiving device for the RS-232. The **RS-423 bus** is similar to the RS-232 in that it is single-ended, but it has a higher data rate and longer cable length. The RS-485 bus is a differential standard and can accommodate multiple drivers and receivers. Table 12–9 summarizes some of the features of the RS (EIA) buses.

TABLE 12–9

Specifications	RS-232	RS-423	RS-422	RS-485
Operation	Single-ended	Single-ended	Differential	Differential
Drivers/Receivers	1/1	1/10	1/10	32/32
Cable length	50 ft	4000 ft	4000 ft	4000 ft
Max data rate	20 kbps	100 kbps	10 Mbps	10 Mbps
Driver output signal level (+/– min/max)	5 V/15 V	3.6 V/6 V	2 V/6 V	1.5 V/6 V

The SPI Bus

The **serial-to-peripheral interface (SPI) bus** is a synchronous serial communications bus that uses four wires for communication between a "master" device and a "slave" device. This standard was developed by Motorola; it operates in full-duplex mode up to a data rate of 10 Mbps and can accommodate multiple slaves. The four signal wires are

1. MOSI (master out slave in) is initiated by the master and received by the slave.
2. MISO (master in slave out) is initiated by the slave and received by the master.
3. SCLK (serial clock) is generated by the master for synchronizing data transfers.
4. SS (slave select) is generated by the master to select an individual slave.

Other names are sometimes assigned to these signals such as SDI (serial data in) for MOSI and SDO (serial data out) for MISO. Figure 12–69 shows a master with a single slave.

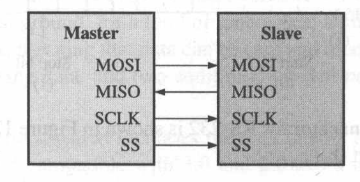


FIGURE 12–69
SPI master/slave configuration.

The SPI bus is typically used in embedded systems and on PCBs for communication between microprocessors or microcontrollers and peripheral IC chips or between two processors. Much PCB "real estate" can be saved compared to using an internal parallel bus with many more connections.

SPI applications include digital audio, signal processing, and telecommunications. SPI is used to communicate with various types of peripherals such as sensors, camera lenses, flash memory, LCD displays, and video games. Many microprocessors and microcontrollers include SPI controllers that can be used as either a master or a slave.

The I²C Bus

I²C bus (pronounced I squared C) is also stated at I2C (I two C) and stands for *inter-integrated circuit.* It is an internal serial bus primarily for connecting ICs on a PCB. A main advantage is that it requires only two lines (plus ground) and therefore saves considerable board space compared to a parallel bus. Two signals (SDA and SCL) are used to communicate between compatible devices. Data are sent serially on the SDA line, and a clock is sent on the SCL line. Like SPI, I²C is based on the master/slave concept where the master

device drives the clock line and the slaves respond to the master. Only the master can initiate a transfer over the bus, but slaves can transfer data under control of the master using clock rates up to 100 kHz in the standard mode. Two other modes, enhanced and high-speed, allow 400 kbps and 3.4 Mbps, respectively.

When transferring data from master to slave, the master device sends a start bit, followed by a slave address, and a write bit. The master waits for an acknowledge (ACK) signal from the slave and then sends the data and waits for an acknowledge before sending a stop bit, as illustrated in Figure 12–70(a). The yellow segments are from the master and the gray elements are from the slave. Similarly, when the master requires data from the slave, it sends a start bit followed by the address and a read bit. The slave returns an acknowledge followed by the data. When the master receives the data, it issues an acknowledge and a stop bit, as shown in Figure 12–70(b).

FIGURE 12–70
I^2C data transfers. Yellow is from master. Gray is from slave.

Start	*Address*	*Write*	*ACK*	*Data*	*ACK*	*Stop*

(a) Data transfer from master to slave

Start	*Address*	*Read*	*ACK*	*Data*	*ACK*	*Stop*

(b) Data transfer from slave to master

The CAN Bus

The **controller area network (CAN) bus**, a differential serial bus, was developed for automotive applications and is also commonly used in aerospace systems, as well as other applications. The bus consists of a terminated twisted pair of signal lines, called CAN H and CAN L, plus ground. Vehicles sold in the United States are required by the SAE (Society of Automotive Engineers) to use the CAN bus protocol. The European Union has similar requirements.

Devices, called *nodes*, can be connected to the bus but are not assigned specific addresses as in the I^2C bus. Two CAN specifications are in use. The standard or basic CAN 2.0A has 11-bit message identifiers and can operate up to 250 kbps, and the full CAN has 29-bit message identifiers and can be used up to 1 Mbps. The message identifier is a label for the contents of a message and goes to each node on the bus. Each receiving node performs a test on the identifier to determine if it is relevant to that node and is used to arbitrate the bus to determine if the message is of highest priority. All of the nodes on the bus can transmit and receive messages. The bus is available to a node with a message with the highest priority (dominant) and can override a message with lower priority (recessive). When the dominant message has been processed, the recessive message is retransmitted.

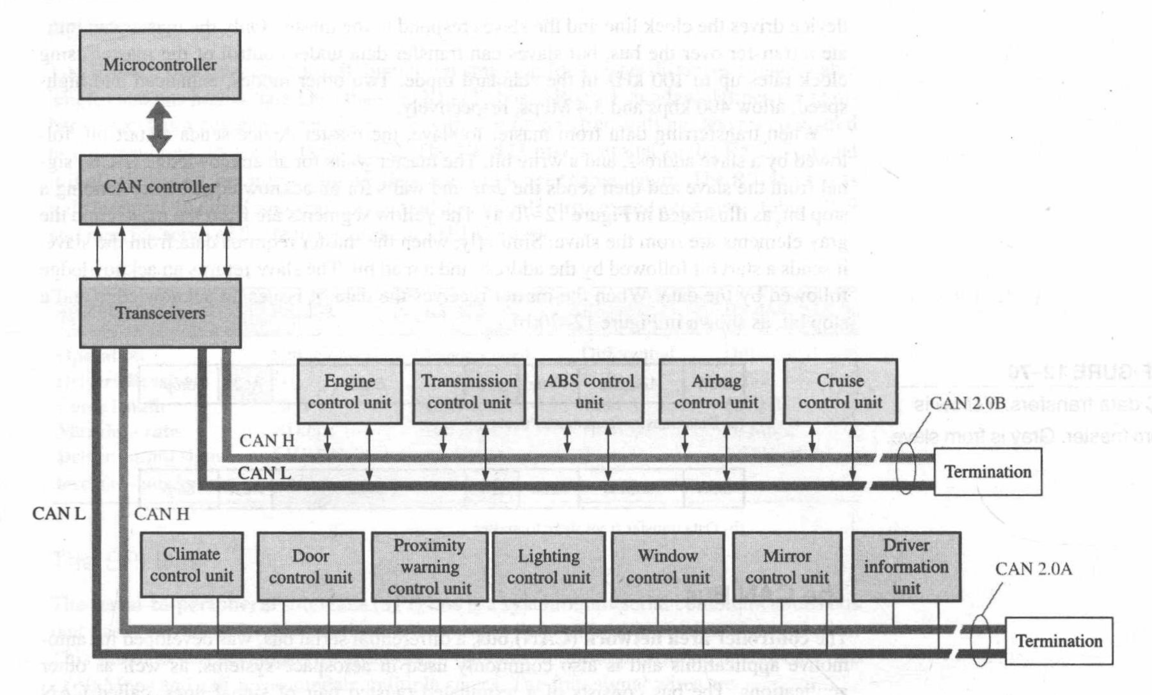
The standard CAN data frame is shown in Figure 12–71. Data are transmitted in NRZ format. The frame begins with a start-of-frame (SOF) bit followed by an arbitration field and a control field. The arbitration field contains the message identifier and a remote transmission request (RTR) bit. The control field has two reserve bits and a data length code (DLC) that specifies the length of the data field that follows and can contain up to 8 bytes. The cyclic redundancy check (CRC) field provides for error detection. The acknowledge (ACK) verifies the receipt of correct data, and the frame ends with the end-of-frame field (EOF).

FIGURE 12–71
Standard CAN data frame format.

SOF	Arbitration field	Control field	Data field	CRC field	ACK	EOF
(1 bit)	Identifier (11 bits) RTR (1 bit)	Reserve (2 bits) DLC (4 bits)	(0–8 bytes)	(16 bits)	(2 bits)	(7 bits)

An Application

An automobile typically has many control units (usually several dozen) for various subsystems, including the engine control unit and other control units for transmission; ABS; cruise control; power steering; audio system; window, door, and mirror controls; airbags; and others. Figure 12–72 is a block diagram of a partial automotive control system using two CAN buses, one low-speed and one high-speed to control various functions throughout the vehicle.



▲ **FIGURE 12–72**

The CAN bus in an automotive control system.

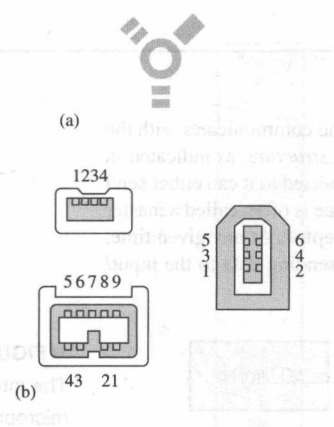
Each unit connected to the bus contains sensors and other functions that allow it to carry out its unique purpose. For example, the ABS (antilock braking system) can receive a message from sensors in each wheel indicating that the brake is about to lock up. A sudden and rapid deceleration in the wheel indicates an imminent lock-up condition. The ABS unit then sends a message that causes the valve in the brake line to release pressure to allow acceleration. Then, when acceleration is sensed, the unit causes a pump to restore the pressure. A rapid release and restore cycle occurs until the brakes are brought under control. A pulsing of the brake pedal can be felt when the operation occurs.

As another example, part of the engine control unit's operation is to sense parameters such as engine temperature, oil pressure, fuel consumption, and rpm, and send messages to the driver unit. All of the units on the bus operate as a system to keep the vehicle running as smoothly and as safely as possible, while providing a comfortable environment for the driver and passengers.

The Firewire Bus

Firewire, also known as IEEE-1394 and iLink, is a high-speed external serial bus developed by Apple Inc. Firewire is used in high-speed communications and real-time data transfer. It is used in professional audio and video equipment, camcorders, DVD players, external hard drives, and in computers used for audio and video editing, as well as in some auto and aircraft applications. It is similar to the USB except that it has a higher data rate and can handle more data.

Three types of connectors are used in the Firewire standard: a 4-pin connector, a 6-pin connector, and a 9-pin connector. The cable for the 4-pin connector consists of two twisted pairs that carry data. The cable for the 6-pin connector has the two twisted pairs for data plus a power line and a ground line. The cable for the 9-pin connector has the same wires as the 6-pin configuration plus two wires that provide for a grounded shield and one wire that is currently unused. The Firewire symbol is shown in Figure 12–73(a). End views of the three connector types are shown in part (b), and the pin designations are shown in part (c).



4-PIN	6-PIN	9-PIN	Description	Wire Color
	1	8	+V dc (30 V max)	White
	2	6	Ground	Black
1	3	1	TPB—(diff signal)	Orange
2	4	2	TPB+	Blue
3	5	3	TPA—(diff signal)	Red
4	6	4	TPA+	Green
		5	Shield	
		7	Unused	
		9	Shield	

▲FIGURE 12–73

Firewire symbol with cable and connector wires and pins.

The Firewire bus address has a total of 64 bits. Ten are for bus ID, six are for node ID, and 48 are for individual addresses. This allows up to 1023 buses, each having up to 63 nodes. The six transfer modes in the IEEE-1394 standard and its revisions are S100, S200, S400, S800, S1600, and S3200. The S100 is the base rate of 98.304 Mbps. The S200 is twice the base rate at 196.608 Mbps, and the S400 is four times the base rate at 393.216 Mbps. The S800 is 786.432 Mbps while the S1600 and S3200 are 16 and 32 times the base rate, respectively (1.6 Gbps and 3.2 Gbps). Firewire cable length cannot exceed 15 ft (4.572 m). To increase this length, up to 16 cables can be connected together.

Firewire versus USB

In general, any capable node can control the bus in a Firewire system, but a single host is used to control the bus in USB. USB networks use a tiered-star topology and Firewire uses a tree topology. A Firewire device can communicate with any node at any time if the conditions allow, but a USB 2.0 device cannot communicate with the host device unless requested by the host. However, USB 3.0 allows Firewire-like communications between devices. USB provides 5 V power while Firewire provides up to 30 V. As a result, Firewire can supply more power to a device than USB. As mentioned before, Firewire is faster than USB.

Serial SCSI

Serial Attached SCSI (**SAS**) is a data-transfer technology for transmitting data to and from storage devices. It has become a replacement for parallel SCSI bus technology that is commonly used in data centers, workstations, and servers. The serial SCSI overcomes some of the limitations of the parallel SCSI. The SAS supports up to a 12 Gbps data rate and allows up to 65,535 devices to be connected using expanders compared to 15 devices for parallel SCSI.

SECTION 12-9 CHECKUP

1. List all the buses introduced in this section.
2. What does SPI stand for?
3. What does I2C stand for?
4. What does CAN stand for?
5. What is another designation for Firewire?

12-10 Bus Interfacing

Basic Multiplexed Buses

As you have learned, in computers the microprocessor controls and communicates with the memories and the input/output (I/O) devices via the *internal bus structure,* as indicated in Figure 12–74. A bus is multiplexed so that any of the devices connected to it can either send or receive data to or from one of the other devices. A sending device is often called a master or **source**, and a receiving device is often called a servant or **acceptor**. At any given time, there is only one source active. For example, the RAM may be sending data to the input/output (I/O) interface under control of the microprocessor.

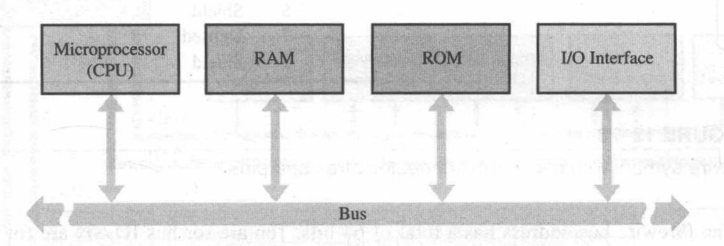


FIGURE 12–74 The interconnection of microprocessor-based system components by a bidirectional, multiplexed bus.

Bus Signals

With synchronous bus control, the microprocessor (CPU) usually originates all control and timing signals. The other devices then synchronize their operations to those control and timing signals. With asynchronous bus control, the control and timing signals are generated jointly by a source and a receiver using a handshaking routine. A typical handshaking sequence is given in Figure 12–75. Handshaking routines may differ from one system to another, as you can see by comparing this sequence with the one shown in Figure 12–53.

An important control function is called **bus arbitration**. Arbitration prevents two sources from trying to use the bus at the same time.

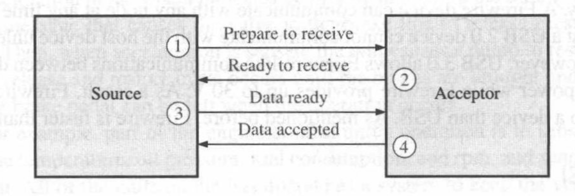


FIGURE 12–75 An example of a handshaking sequence.

Connecting Devices to a Bus

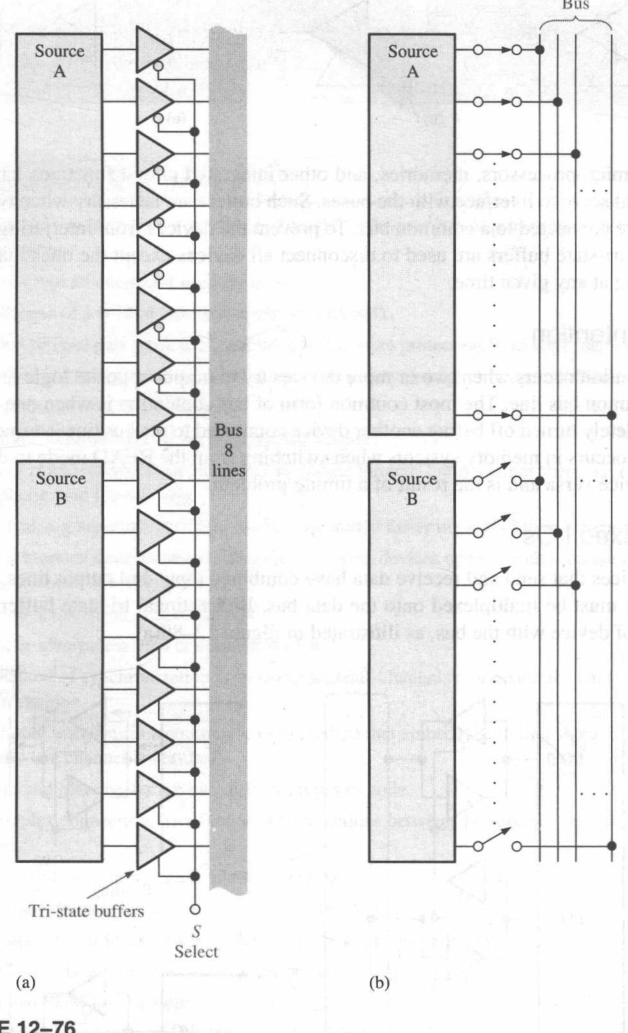
Tri-state buffers are normally used to interface the outputs of a source device to a bus. Usually more than one source is connected to a bus, but only one can have access at any given time. All the other sources must be disconnected from the bus to prevent **bus contention**.

Tri-state circuits are used to connect a source to a bus or disconnect it from a bus, as illustrated in Figure 12–76(a) for the case of two sources. The select input is used to connect either source A or source B but not both at the same time to the bus. When the select input is LOW, source A is connected and source B is disconnected. When the select input is HIGH, source B is connected and source A is disconnected. A switch equivalent of this action is shown in part (b) of the figure.

When the enable input of a tri-state circuit is not active, the device is in a high-impedance (**high-Z**) state and acts like an open switch. Many digital ICs provide internal tri-state buffers for the output lines. A tri-state output is indicated by a ∇ symbol as shown in Figure 12–77.

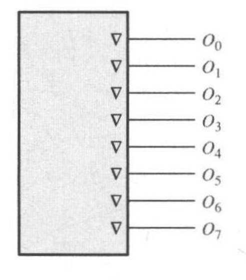
Tri-State Buffer Operation

Figure 12–78(a) shows the logic symbol for a noninverting tri-state buffer with an active-HIGH enable. Part (b) of the figure shows one with an active-LOW enable.



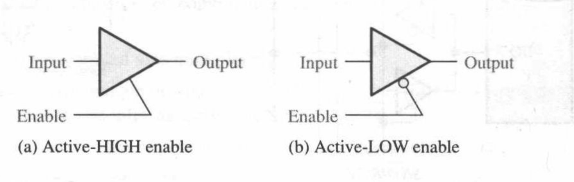
▲**FIGURE 12–76**

Tri-state buffer interface to a bus.



▲**FIGURE 12–77**

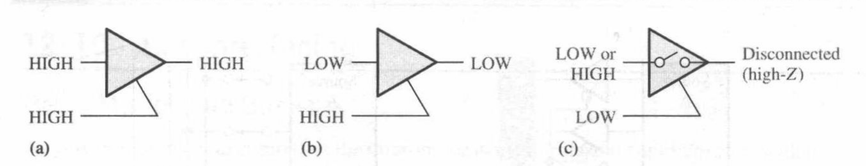
Method of indicating tri-state outputs on an IC device.



▲**FIGURE 12–78**

Tri-state buffer symbols.

The basic operation of a tri-state buffer can be understood in terms of switching action as illustrated in Figure 12–79. When the enable input is active, the gate operates as a normal noninverting circuit. That is, the output is HIGH when the input is HIGH and LOW when the input is LOW, as shown in parts (a) and (b) respectively. The HIGH and LOW levels represent two of the states. The buffer operates in its third state when the enable input is not active. In this state, the circuit acts as an open switch, and the output is completely disconnected from the input, as shown in part (c). This is sometimes called the *high-impedance* or *high-Z* state.



◄ **FIGURE 12–79**
Tri-state buffer operation.

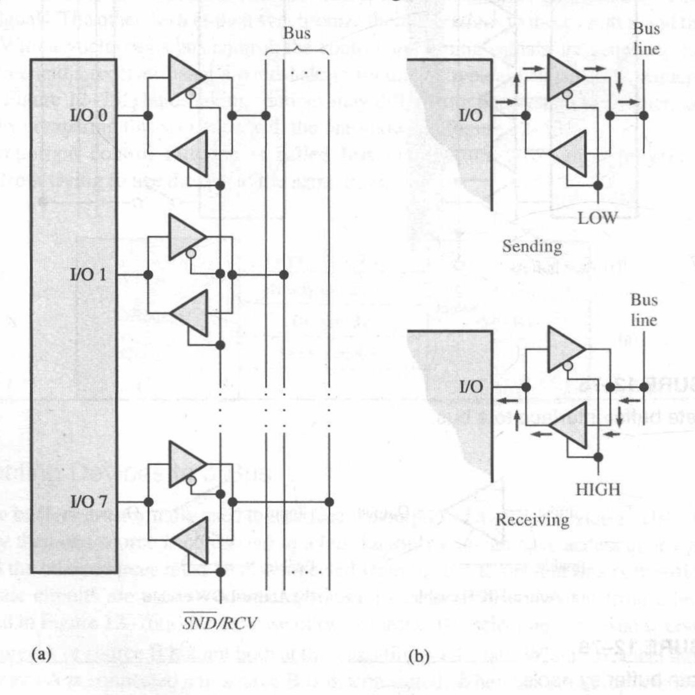
Many microprocessors, memories, and other integrated circuit functions have tri-state buffers that serve to interface with the buses. Such buffers are necessary when two or more devices are connected to a common bus. To prevent the devices from interfering with each other, the tri-state buffers are used to disconnect all devices except the ones that are communicating at any given time.

Bus Contention

Bus contention occurs when two or more devices try to output opposite logic levels on the same common bus line. The most common form of bus contention is when one device has not completely turned off before another device connected to the bus line is turned on. This generally occurs in memory systems when switching from the READ mode to the WRITE mode or vice versa and is the result of a timing problem.

Multiplexed I/Os

Some devices that send and receive data have combined input and output lines, called I/O ports, that must be multiplexed onto the data bus. Bidirectional tri-state buffers interface this type of device with the bus, as illustrated in Figure 12–80(a).



◄ **FIGURE 12–80**
Multiplexed I/O operation.

Each I/O port has a pair of tri-state buffers. When the $\overline{SND}/RCV$ ($\overline{\text{Send}}$/Receive) line is LOW, the upper tri-state buffer in each pair is enabled and the lower one disabled. In this state, the device is acting as a source and sending data to the bus. When the $\overline{SND}/RCV$ line is HIGH, the lower tri-state buffer in each pair is enabled so that the device is acting as an acceptor and receiving data from the bus. This operation is illustrated in Figure 12–80(b). Some devices provide for multiplexed I/O operation with internal circuitry.

SECTION 12-10 CHECKUP

1. Why are tri-state buffers required to interface digital devices to a bus?
2. What is the purpose of a bus system?

TRUE/FALSE QUIZ

Answers are at the end of the chapter.

1. The simplest connection for sending and receiving devices in a data transmission system is a coaxial cable.
2. BNC is a type of connector used for coax.
3. The purpose of a twisted pair is to minimize crosstalk.
4. Shielded twisted pair cable in a conduit provides more protection from EMI than UTP.
5. Fiber-optic cables are used to transmit electrical pulses through thin optical fibers.
6. ST is a type of fiber-optic connector.
7. Most data communications occur within the visible spectrum.
8. Types of rf and microwave signals that propagate through Earth's atmosphere are ground wave, ionospheric, and line-of-sight.
9. In general, a given number of bits can be transmitted faster in parallel than in series.
10. In asynchronous systems, the sending and receiving devices operate with separate oscillators having different clock frequencies.
11. Data rate is the speed of data transfer.
12. Bit rate is always less than or equal to the baud.
13. One method of synchronization is by using separate channels to transmit the data and the timing information.
14. Manchester encoding is a commonly used method that embeds the timing signal in the data so that only one channel is required.
15. Biphase and Manchester are two different types of code.
16. In a simplex connection, data flow in both directions between the sender (transmitter) and the receiver.
17. BASK stands for binary amplitude-shift keying.
18. PSK applications include wireless LAN and bluetooth.
19. In quadrature amplitude modulation there are eight phase quadrants.
20. QAM is widely used in telecommunications and in digital cable TV.
21. PWM and PDM are the same.
22. PPM is less sensitive to channel interference than PAM or PWM.
23. Pulse code modulation involves sampling of an analog signal amplitude at regular intervals.
24. The three main data transmission combinations are digital-to-analog, analog-to-digital, and digital-to-digital.
25. TDM stands for transmitted data multiplexing.
26. FDM is a baseband technique in which the total bandwidth available to a system is divided into frequency sub-bands and information is sent in analog form.
27. A bus connects two or more devices to allow them to communicate.
28. A bus is only defined by the wires and connectors.
29. A parallel bus is always faster than a serial bus.
30. Bus width is the width of each conductor in a parallel bus.
31. Handshaking is part of bus protocol.
32. A single-ended transmission is simpler and lower in cost compared to a differential transmission.
33. A tri-state driver has a HIGH state, a LOW state, and a shorted state.
34. PCI stands for *peripheral computer interface*.
35. Two types of PCI buses are PCI-X and PCI-E.
36. IEEE-488 is known as the GPIB.
37. SCSI stands for *serial computer system interface*.

38. USB is a widely used standard serial bus for connecting peripherals to a computer.
39. The SPI bus is typically used in embedded systems and on PCBs for communications between microprocessors or microcontrollers and peripheral IC chips or between two processors.
40. An internal serial bus primarily for connecting ICs on a PCB is the I^2C bus.
41. Most automotive systems use the Firewire bus.
42. CAN stands for *computer area node*.

SELF-TEST

Answers are at the end of the chapter.

1. The main purpose of the shield in a coaxial cable is to
   (a) make the cable stronger (b) dissipate heat
   (c) prevent EMI (d) prevent distortion
2. UTP is color-coded according to
   (a) a 10-pair color code (b) a 25-pair color code
   (c) the resistor color code (d) the primary colors
3. Advantages that fiber-optic systems have over electrical transmission media are
   (a) higher data rate, less susceptible to noise, and longer transmission distance
   (b) lower cost, higher data rate, and simplicity
   (c) higher data rate, higher EMI, less distortion
   (d) higher baud, availability, and reliability
4. The modes of light propagation in optical fibers are
   (a) simplex and duplex (b) multimode and single-mode
   (c) synchronous and asynchronous (d) scatter and direct
5. The electromagnetic spectrum includes
   (a) radio waves, microwaves, and audio waves
   (b) radio waves, microwaves, and sonic waves
   (c) visible light, infrared, and alpha waves
   (d) radio waves, microwave, and infrared
6. Satellites use
   (a) ground propagation (b) line-of-sight propagation
   (c) ionospheric propagation (d) triangulation
7. In asynchronous transmission, data are sent in short bursts known as
   (a) packets (b) frames
   (c) bundles (d) quanta
8. For a given bit rate, the baud compared to the bit rate is
   (a) always greater (b) always less
   (c) equal or greater (d) equal or less
9. The efficiency of a data transmission system is
   (a) the ratio of baud to bit rate
   (b) the ratio of actual data rate to ideal data rate
   (c) the ratio of data bits to total bits
   (d) the ratio of parity bits to data bits
10. The Manchester code format is
    (a) NRZ (b) biphase
    (c) RZ (d) FDM
11. A synchronous data frame does not contain a(n)
    (a) preamble (b) data field
    (c) address (d) vector field
12. Three types of data channel connections in terms of data flow are
    (a) input, output, neutral
    (b) simplex, half-duplex, full-duplex
    (c) simplex, duplex, triplex
    (d) uniplex, diplex, biplex
13. In FSK modulation,
    (a) the frequency of a carrier signal is varied by a digital signal.
    (b) the frequency of a digital signal is varied by a carrier signal.
    (c) the phase of a carrier signal is varied by a digital signal.
    (d) the amplitude of a carrier signal is varied by a digital signal.

14. Types of modulation in which a parameter of a sine-wave carrier signal is varied by a digital signal are
(a) PAM, PWM, PPM **(b)** QAM, PAM, ASK
(c) FSK, PSK, PPM **(d)** FSK, ASK, PSK

15. QAM stands for
(a) quadrature analysis method
(b) quadrature amplitude modulation
(c) quasi-amplitude modulation
(d) quadratic amplitude modulation

16. In QAM, the parameters that are varied are
(a) amplitude and frequency **(b)** phase and frequency
(c) amplitude and phase **(d)** pulse width and position

17. Three methods of modulating a digital signal with analog data are
(a) PAM, PWM, PPM **(b)** PAM, ASK, PPM
(c) FSK, QAM, PAM **(d)** QAM, PAM, PWM

18. The most likely type of modulation to be used in motor speed control is
(a) PAM **(b)** PPM
(c) PWM **(d)** QAM

19. A method in which an analog signal is sampled and converted to a digital code is
(a) PCM **(b)** PDM
(c) PDC **(d)** PPM

20. TDM stands for
(a) time duration modulation **(b)** time division modulation
(c) time division multiplexing **(d)** time division method

21. Two methods of combining data in TDM are
(a) fast and slow **(b)** bit-interleaved and byte-interleaved
(c) time-domain and frequency-domain **(d)** simplex and duplex

22. A method by which data from multiple sources are sent simultaneously is
(a) TDM **(b)** FCM
(c) FSK **(d)** FDM

23. Properties that define a bus include
(a) type of connectors
(b) length and type of cable or connection
(c) data rate and encoding
(d) all of these

24. The speed of a parallel bus can be limited by
(a) crosstalk **(b)** EMI
(c) skew **(d)** all of these

25. The method by which two devices initiate and complete a bus transfer is
(a) handshaking **(b)** saluting
(c) give and take **(d)** multiple access protocol

26. PCI is the acronym for
(a) peripheral controller interface **(b)** peripheral computer interface
(c) protocol compatible interface **(d)** peripheral component interconnect

27. In a PCI system, the individual paths from switch to peripherals are called
(a) pipes **(b)** lanes
(c) highways **(d)** channels

28. The following is not a classification of USB:
(a) low-speed **(b)** full-speed
(c) high-speed **(d)** intermediate speed

29. A 3.0 USB cable contains
(a) two twisted pairs **(b)** one twisted pair
(c) three twisted pairs **(d)** two straight wires

30. Four types of USB packets are
(a) token, data, handshake, and start-of-frame
(b) token, data, handshake, and control
(c) identification, address, synchronization, and data
(d) none of these

31. The RS-232 encoding method is
(a) RZ (b) Manchester
(c) biphase (d) NRZ

32. The bus typically used to connect systems in an automobile is the
(a) SPI (b) CAN
(c) I^2C (d) PCI

PROBLEMS

Answers to odd-numbered problems are at the end of the book.

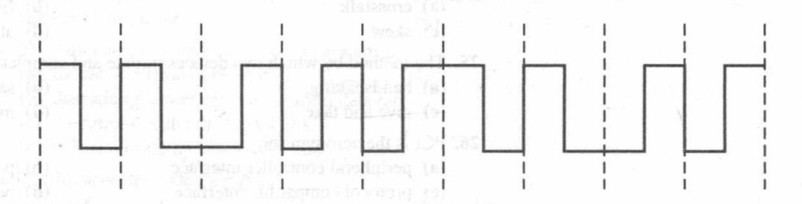
Section 12–1 Data Transmission Media

1. List four parts of a coaxial cable.
2. What do the acronyms UTP and STP mean?
3. Name four advantages of fiber optic media over electrical transmission media.
4. List three parts of an optical fiber.
5. Describe the multimode in an optical fiber.
6. Which of the optical fiber sizes (50/125, 62.5/125, 8.3/125) operates in multimode and which operate(s) in single mode?
7. Draw a basic block diagram of a fiber optics communications link.
8. A frequency of 100 MHz falls into what part of the electromagnetic spectrum?
9. In what frequency range does visible light fall?

Section 12–2 Methods and Modes of Data Transmission

10. If data bits are transmitted serially at a 1 MHz rate, how many bits can be transmitted in 1 ms?
11. If a byte of data is transmitted in parallel in 1 μs, what is the data rate in bits per second?
12. Eight voltage levels are being transmitted by a system where each level (symbol) represents a 3-bit code. Assuming that 12 symbols are transmitted in 0.5 μs, determine the bit rate and the baud.
13. Assume a 5-bit code is used for each symbol transmitted. If the bit rate is 25 MHz, what is the baud?
14. A certain data packet contains a total of 20 bits of which 16 are data bits. Determine the efficiency.
15. Show the data waveform for the bit sequence 101011100011 in NRZ and in RZ formats.
16. For the bit sequence in Problem 15, show the Manchester code.
17. Determine the bit sequence represented by the Manchester code in Figure 12–81.

FIGURE 12–81

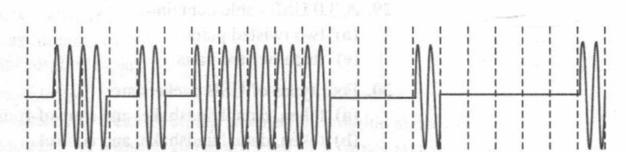


18. List and describe each part of a typical synchronous data frame structure.

Section 12–3 Modulation of Analog Signals with Digital Data

19. Determine the binary code represented by the ASK signal in Figure 12–82. Presence of a signal is a 1 and absence of a signal is 0.

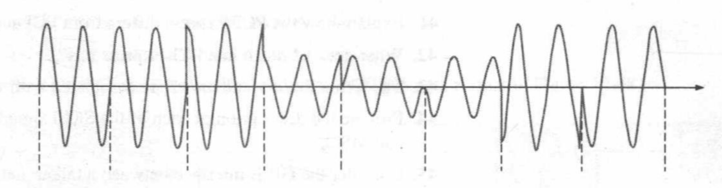
FIGURE 12–82



20. Show how you would respresent four successive bits (1010) using FSK.
21. Repeat Problem 20 for PSK.

22. Refer to Figure 12–23 and determine the sequence of bits represented by the QAM signal in Figure 12–83.

▶ FIGURE 12–83

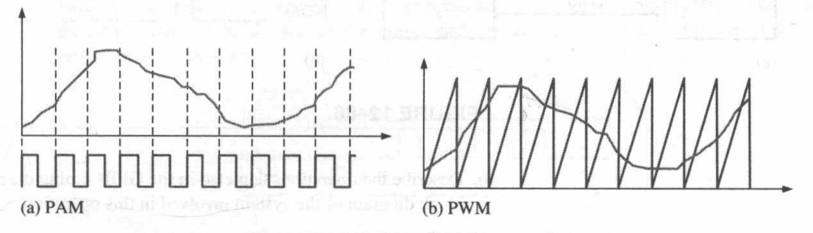


23. Sketch a constellation map for a 4-bit PSK system.
24. Repeat Problem 23 for a 4-bit QAM system.

Section 12–4 Modulation of Digital Signals with Analog Data

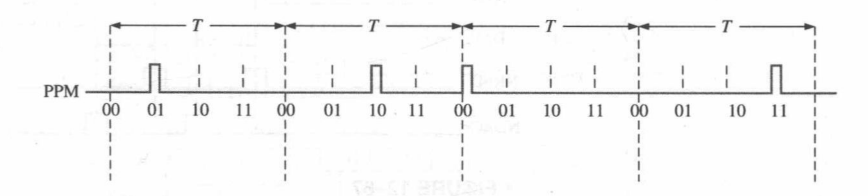
25. Describe the PWM intersective method.
26. Develop the PAM and PWM signals for the waveform in Figure 12–84.

▶ FIGURE 12–84



27. In a certain PPM system there are 2^3 positions in each period, T. If $T = 10$ ms, determine the data rate.
28. Show the NRZ code for the PPM signal in Figure 12–85.

▶ FIGURE 12–85



29. In a PCM code, how many bits are required to represent 16 voltage levels of a modulating signal?
30. Show the 4-bit PCM code in NRZ format for four successive samples of an analog waveform. The sampled values are 1, 3, 5, and 7.

Section 12–5 Multiplexing and Demultiplexing

31. Explain the difference between bit-interleaved and byte-interleaved TDM.
32. Describe the difference between synchronous and statistical TDM.
33. What type of filters are used in an FDM system and what is their purpose?
34. What is the frequency separation called between each source in an FDM system?

Section 12–6 Bus Basics

35. List six physical characteristics and five electrical and performance characteristics of a bus.
36. Explain the difference between bus width and bus bandwidth.
37. A certain bus is specified with a width of 16 bits and a frequency of 100 MHz. Determine the bus bandwidth expressed as two different values.
38. Describe a simple example of a handshake.
39. State an advantage of a differential bus over a single-ended bus.

Section 12–7 Parallel Buses

40. Explain the difference between PCI and PCI-X buses.

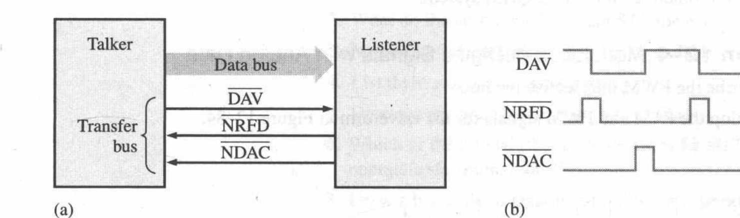
41. Explain how the PCI-Express differs from PCI and PCI-X.

42. What does x2 mean in a PCI-Express bus?

43. The terms *listener* and *talker* are associated with which bus standard?

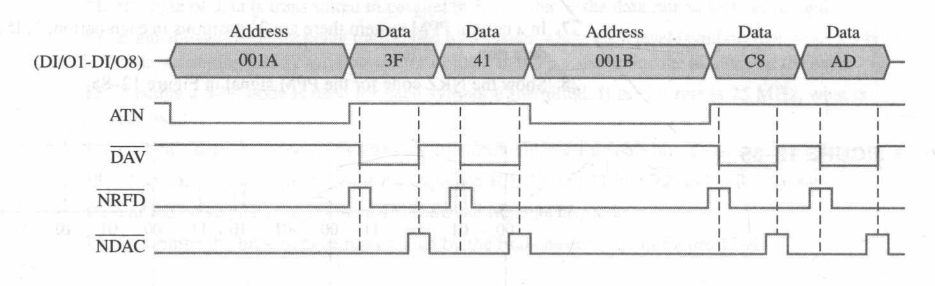
44. Provide the description of each of the SCSI signals: BSY, SEL, RST, C/D, REQ, ACK, ATN, and MSG.

45. Consider the GPIB interface between a talker and a listener as shown in Figure 12–86(a). From the handshaking timing diagram in part (b), determine how many data bytes are actually transferred to the listening device.



▲ FIGURE 12–86

46. Describe the operations depicted in the GPIB timing diagram of Figure 12–87. Develop a basic block diagram of the system involved in this operation.

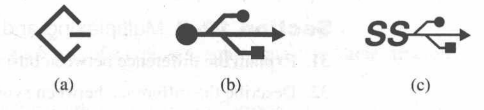


▲ FIGURE 12–87

Section 12–8 The Universal Serial Bus (USB)

47. Identify each of the symbols in Figure 12–88.

► FIGURE 12–88



48. List the four types of USB packets.

49. Describe each of the fields in the USB data packet in Figure 12–89.

► FIGURE 12–89

Sync	PID	Data	CRC	EOP
8/32	8	0-8192	16	3

50. What type of data encoding is used in USB 3.0?

51. Determine the maximum number of bytes in a USB data field.

52. What is the maximum separation of two USB 2.0 devices?

Section 12–9 Other Serial Buses

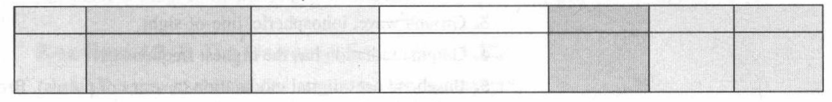
53. Describe how RS-232 and RS-422 differ.

54. List the four signals in an SPI bus and describe each one.

55. Describe the main use for the I^2C bus.

56. Fill in the field descriptions for the blank CAN bus data format in Figure 12–90.

▶ FIGURE 12–90



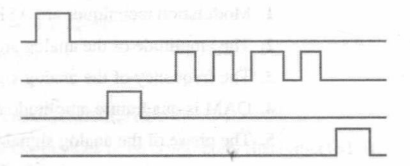
57. Refer to Figure 12–72 and list additional units that may appear on a CAN automotive system.

58. What is the data rate for the IEEE-1394 bus standard in the S100 mode? What is the data rate in the S1600 mode?

Section 12–10 Bus Interfacing

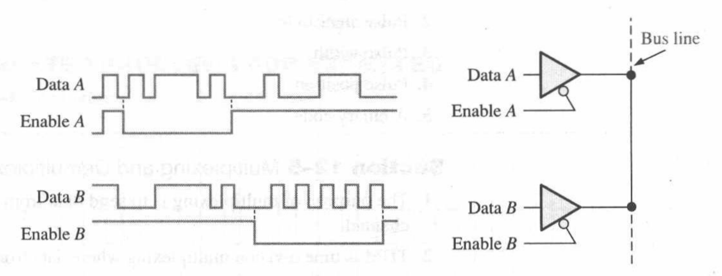
59. In a simple serial transfer of eight data bits from a sending device to an receiving device, the handshaking sequence in Figure 12–91 is observed on the four generic bus lines. By analyzing the time relationships, identify the function of each signal and indicate if it originates at the sender or at the receiver.

▶ FIGURE 12–91

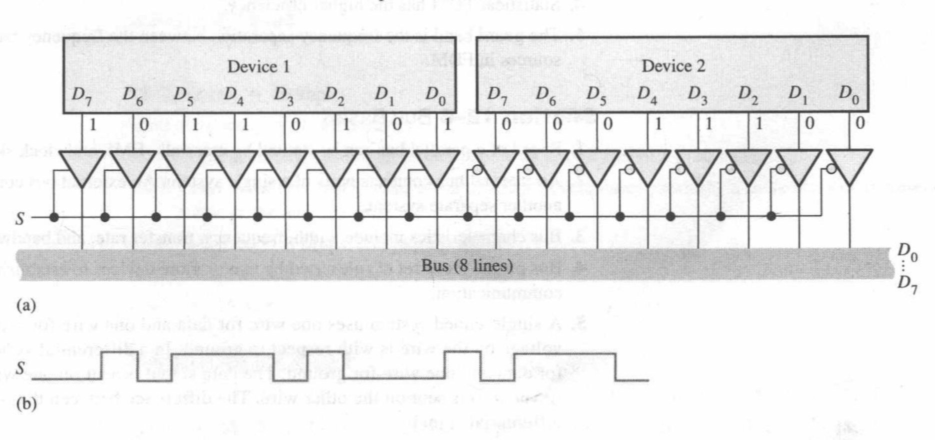


60. Determine the signal on the bus line in Figure 12–92 for the data input and enable waveforms shown.

▶ FIGURE 12–92



61. In Figure 12–93(a), data from the two sources are being placed on the data bus under control of the select line. The select waveform is shown in Figure 12–93(b). Determine the data bus waveforms for the device output codes indicated.



▲ FIGURE 12–93

ANSWERS

SECTION CHECKUPS

Section 12–1 Data Transmission Media

1. Wire, coaxial cable, twisted pair cable, optical fiber cable, and wireless
2. The shield protects against EMI.
3. Ground wave, ionospheric, line-of-sight
4. Gamma radiation has the highest frequencies
5. Baseband uses digital modulation (a series of pulses). Broadband uses a digitally modulated analog signal.

Section 12–2 Methods and Modes of Data Transmission

1. Serial data are one bit at a time in sequence. Parallel data are simultaneous multiple bits at a time.
2. Synchronization allows the receiver to recognize the beginning and end of a data transmission.
3. RZ, NRZ, biphase are three types of data format.
4. Simplex, half-duplex, and full-duplex are three modes of data transmission.

Section 12–3 Modulation of Analog Signals with Digital Data

1. Modulation techniques are ASK, FSK, PSK, and QAM.
2. The amplitude of the analog signal is changed in ASK.
3. The frequency of the analog signal is changed in FSK.
4. QAM is quadrature amplitude modulation.
5. The phase of the analog signal is changed in PSK.

Section 12–4 Modulation of Digital Signals with Analog Data

1. Pulse modulation methods are PAM, PWM, PPM, and PCM
2. Pulse amplitude
3. Pulse width
4. Pulse position
5. A binary code

Section 12–5 Multiplexing and Demultiplexing

1. The purpose of multiplexing is to send data from several sources on a single communication channel.
2. TDM is time division multiplexing where data from multiple sources are interleaved on a time basis.
3. FDM is frequency division multiplexing where data from multiple sources are sent simultaneously at different frequencies.
4. Statistical TDM has the higher efficiency.
5. The guard band is the frequency separation between the frequency bands of the multiple sources in FDM.

Section 12–6 Bus Basics

1. Speed of a parallel bus can be limited by crosstalk, EMI, and clock skew.
2. An internal bus connects parts of a single system. An external bus connects one system to another separate system.
3. Bus characteristics include width, frequency, transfer rate, and bandwidth.
4. Bus protocol is a set of rules used by two or more devices to establish and maintain communication.
5. A single-ended system uses one wire for data and one wire for ground, where the signal voltage on the wire is with respect to ground. In a differential system, two wires are used for data and one wire for ground. The data signal is sent on one wire and its complement (inversion) is sent on the other wire. The difference between the two data wires is the differential signal.

Section 12–7 Parallel Buses

1. PCI is peripheral component interconnect.
2. PCI-Express is also designated PCIe and PCI-E.
3. A lane is a dedicated path to a single chip known as a switch.
4. GPIB is IEEE-488.
5. SCSI is small computer system interface.

Section 12–8 The Universal Serial Bus (USB)

1. USB is universal serial bus, a widely used standard bus.
2. USB pins are D+, D–, +5 V, ground.
3. The twisted pair reduces or eliminates noise.
4. USB 3.0 can run at higher speeds than USB 2.0. USB 3.0 has shorter cable lengths than USB 2.0.

Section 12–9 Other Serial Buses

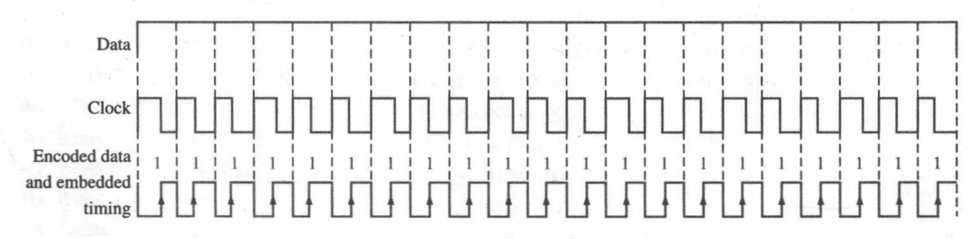
1. RS-232, RS-422, RS-423, RS-495, SPI, I^{2}C, CAN, Firewire, and serial SCSI
2. SPI is serial-to-peripheral interface.
3. I^{2}C is inter-integrated circuit.
4. CAN is controller area network.
5. Firewire is IEEE-1394.

Section 12–10 Bus Interfacing

1. Tri-state buffers allow devices to be completely disconnected from the bus when not in use, thus preventing interference with other devices.
2. A bus interconnects all the devices in a system and makes communication between devices possible.

RELATED PROBLEMS FOR EXAMPLES

12–1 See Figure 12–94.



▲ **FIGURE 12–94**

12–2 Bit rate = 40 kbps

12–3 Efficiency = 0.828 (82.8%)

12–4 Eight amplitudes and eight phases can be used to represent a 4-bit code.

12–5 There would be more pulses closer together in both cases providing a more accurate representation of the analog signal.

12–6 Data rate = 8 Mbps; 256 pulse positions

12–7 Six PCM code bits to represent 64 levels

12–8 15.625 MHz

TRUE/FALSE QUIZ

1. F **2.** T **3.** T **4.** T **5.** F **6.** T **7.** F **8.** T **9.** T
10. F **11.** T **12.** F **13.** T **14.** T **15.** F **16.** F **17.** T **18.** T

19. F **20.** T **21.** T **22.** T **23.** T **24.** T **25.** F **26.** F **27.** T
28. F **29.** F **30.** F **31.** T **32.** T **33.** F **34.** F **35.** T **36.** T
37. F **38.** T **39.** T **40.** T **41.** F **42.** F

SELF-TEST

1. (c) **2.** (b) **3.** (a) **4.** (b) **5.** (d) **6.** (b) **7.** (a) **8.** (d)
9. (c) **10.** (b) **11.** (d) **12.** (b) **13.** (a) **14.** (d) **15.** (b) **16.** (c)
17. (a) **18.** (c) **19.** (a) **20.** (c) **21.** (b) **22.** (d) **23.** (d) **24.** (d)
25. (a) **26.** (d) **27.** (b) **28.** (d) **29.** (c) **30.** (a) **31.** (d) **32.** (b)

Answers to Odd–Numbered Problems

Chapter 1

1. Digital can be transmitted and stored more efficiently and reliably.

3. Clock
Thermometer
Speedometer

5. **(a)** 11010001 **(b)** 000101010

7. **(a)** 550 ns **(b)** 600 ns **(c)** 2.7 μs **(d)** 10 V

9. 250 Hz

11. 50%

13. 8 μs; 1 μs

16. DIP pins go through holes in a circuit board. SMT pins connect to surface pads.

Chapter 2

1. **(a)** 1 **(b)** 100 **(c)** 100,000

3. **(a)** 400; 70; 1 **(b)** 9000; 300; 50; 6
(c) 100,000; 20,000; 5000; 0; 0; 0

5. **(a)** 3 **(b)** 4 **(c)** 7 **(d)** 8 **(e)** 9
(f) 12 **(g)** 11 **(h)** (15)

7. **(a)** 51.75 **(b)** 42.25 **(c)** 65.875
(d) 120.625 **(e)** 92.65625 **(f)** 113.0625
(g) 90.625 **(h)** 127.96875

9. **(a)** 5 bits **(b)** 6 bits **(c)** 6 bits
(d) 7 bits **(e)** 7 bits **(f)** 7 bits
(g) 8 bits **(h)** 8 bits

11. **(a)** 1010 **(b)** 10001 **(c)** 11000
(d) 110000 **(e)** 111101 **(f)** 1011101
(g) 1111101 **(h)** 10111010

13. **(a)** 1111 **(b)** 10101 **(c)** 11100
(d) 100010 **(e)** 101000 **(f)** 111011
(g) 1000001 **(h)** 1001001

15. **(a)** 100 **(b)** 100 **(c)** 1000
(d) 1101 **(e)** 1110 **(f)** 11000

17. **(a)** 1001 **(b)** 1000 **(c)** 100011
(d) 110110 **(e)** 10101001 **(f)** 10110110

19. all 0s or all 1s

21. **(a)** 010 **(b)** 001 **(c)** 0101
(d) 00101000 **(e)** 0001010 **(f)** 11110

23. **(a)** 00011101 **(b)** 11010101
(c) 01100100 **(d)** 11111011

25. **(a)** 00001100 **(b)** 10111100
(c) 01100101 **(d)** 10000011

27. **(a)** −102 **(b)** +116 **(c)** −64

29. **(a)** 0 10001101 11110000101011000000000
(b) 1 10001010 11000001100000000000000

31. **(a)** 00110000 **(b)** 00011101
(c) 11101011 **(d)** 100111110

33. **(a)** 11000101 **(b)** 11000000

35. 100111001010

37. **(a)** 00111000 **(b)** 01011001 **(c)** 101000010100
(d) 010111001000 **(e)** 0100000100000000
(f) 1111101100010111 **(g)** 1000101010011101

39. **(a)** 35 **(b)** 146 **(c)** 26 **(d)** 141
(e) 243 **(f)** 235 **(g)** 1474 **(h)** 1792

41. **(a)** 60_{16} **(b)** $10B_{16}$ **(c)** $1BA_{16}$

43. **(a)** 10 **(b)** 23 **(c)** 46 **(d)** 52 **(e)** 67
(f) 367 **(g)** 115 **(h)** 532 **(i)** 4085

45. **(a)** 001011 **(b)** 101111 **(c)** 001000001
(d) 011010001 **(e)** 101100000
(f) 100110101011 **(g)** 001011010111001
(h) 100101110000000 **(i)** 001000000010001011

47. **(a)** 00010000 **(b)** 00010011
(c) 00011000 **(d)** 00100001
(e) 00100101 **(f)** 00110110
(g) 01000100 **(h)** 01010111
(i) 01101001 **(j)** 10011000
(k) 000100100101 **(l)** 000101010110

49. **(a)** 000100000100 **(b)** 000100101000
(c) 000100110010 **(d)** 000101010000
(e) 000110000110 **(f)** 001000010000
(g) 001101011001 **(h)** 010101000111
(i) 0001000001010001

51. **(a)** 80 **(b)** 237 **(c)** 346 **(d)** 421 **(e)** 754
(f) 800 **(g)** 978 **(h)** 1683 **(i)** 9018 **(j)** 6667

53. **(a)** 00010100 **(b)** 00010010
(c) 00010111 **(d)** 00010110
(e) 01010010 **(f)** 000100001001
(g) 000110010101 **(h)** 0001001001101001

55. The Gray code makes only one bit change at a time when going from one number in the sequence to the next.

57. **(a)** 1100 **(b)** 00011 **(c)** 10000011110

58. **(b)** is incorrect.

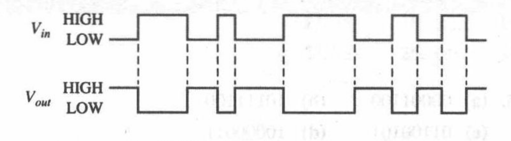
60. (a) 110100100 (b) 000001001 (c) 111111110

62. In each case, you get the other number.

64. The remainder is 0100, indicating an error.

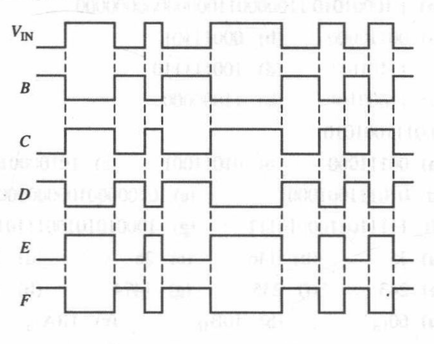
Chapter 3

1. See Figure P–1.



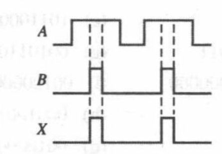
▲FIGURE P–1

3. See Figure P–2.



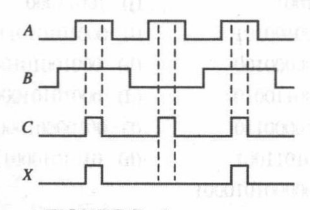
▲FIGURE P–2

5. See Figure P–3.



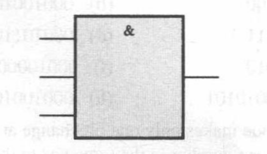
▲FIGURE P–3

7. See Figure P–4.



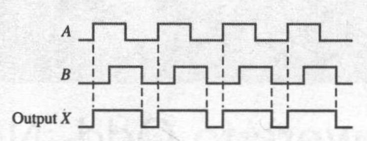
▲FIGURE P–4

9. See Figure P–5.



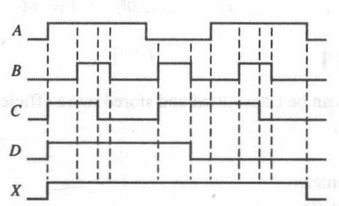
▲FIGURE P–5

11. See Figure P–6.



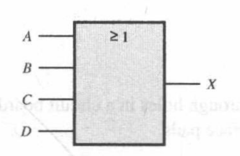
▲FIGURE P–6

13. See Figure P–7.



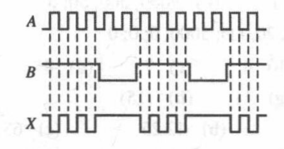
▲FIGURE P–7

15. See Figure P–8.



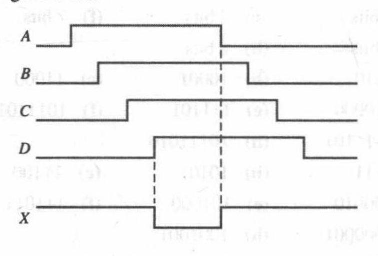
▲FIGURE P–8

17. See Figure P–9.



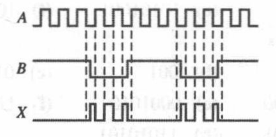
▲FIGURE P–9

19. See Figure P–10.



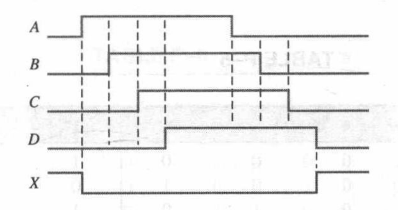
▲FIGURE P–10

21. See Figure P–11.



▲FIGURE P–11

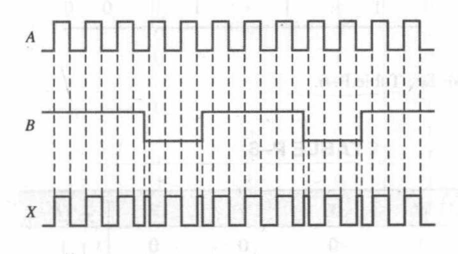
23. See Figure P–12.



▲ **FIGURE P–12**

25. XOR $= A\overline{B} + \overline{A}B$; OR $= A + B$

27. See Figure P–13.



▲ **FIGURE P–13**

29. CMOS

31. $t_{PLH} = 4.3$ ns; $t_{PHL} = 10.5$ ns

33. 20 mW

Chapter 4

1. $X = A + B + C + D$

3. $X = \overline{A} + \overline{B} + \overline{C}$

5. **(a)** $AB = 1$ when $A = 1, B = 1$

(b) $A\overline{B}C = 1$ when $A = 1, B = 0, C = 1$

(c) $A + B = 0$ when $A = 0, B = 0$

(d) $\overline{A} + B + \overline{C} = 0$ when $A = 1, B = 0, C = 1$

(e) $\overline{A} + \overline{B} + C = 0$ when $A = 1, B = 1, C = 0$

(f) $\overline{A} + B = 0$ when $A = 1, B = 0$

(g) $A\overline{B}\,\overline{C} = 1$ when $A = 1, B = 0, C = 0$

7. **(a)** Commutative

(b) Commutative

(c) Distributive

9. **(a)** $\overline{A}B$ **(b)** $A + \overline{B}$

(c) $\overline{A}\,\overline{B}\,\overline{C}$ **(d)** $\overline{A} + \overline{B} + \overline{C}$

(e) $\overline{A} + \overline{B}\,\overline{C}$ **(f)** $\overline{A} + \overline{B} + \overline{C} + \overline{D}$

(g) $(\overline{A} + \overline{B})(\overline{C} + \overline{D})$ **(h)** $\overline{A}B + C\overline{D}$

11. **(a)** $(\overline{A} + \overline{B} + \overline{C})(\overline{E} + \overline{F} + \overline{G})(\overline{H} + \overline{I} + \overline{J})(\overline{K} + \overline{L} + \overline{M})$

(b) $\overline{A}B\overline{C} + BC$

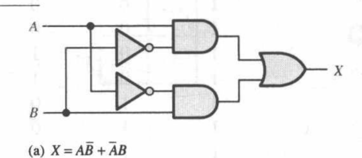
(c) $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}\,\overline{E}\,\overline{F}\,\overline{G}\,\overline{H}$

13. **(a)** $X = ABCD$ **(b)** $X = AB + C$

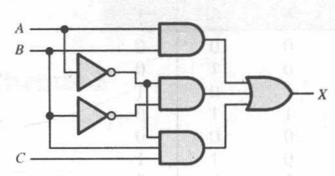
(c) $X = \overline{\overline{A}B}$ **(d)** $X = (A + B)C$

15. See Figure P–14.

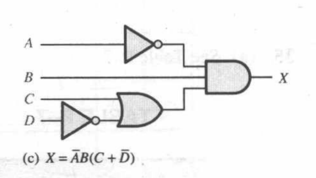
► **FIGURE P–14**



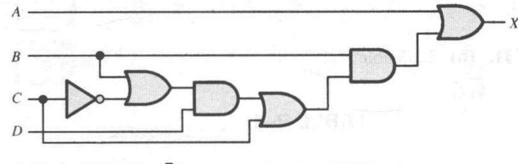
(a) $X = A\overline{B} + \overline{A}B$



(b) $X = AB + \overline{A}\overline{B} + \overline{A}BC$



(c) $X = \overline{A}B(C + \overline{D})$



(d) $X = A + B[C + D(B + \overline{C})]$

17. **(a)** See Table P–1.

▼ **TABLE P–1**

Inputs			Output
$\overline{VCR}$	$\overline{CAMI}$	$\overline{RDY}$	$\overline{RECORD}$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

(b) See Table P–2.

▼ **TABLE P–2**

Inputs			Output
$\overline{RTS}$	$\overline{ENABLE}$	$\overline{BUSY}$	$\overline{SEND}$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

19. **(a)** A **(b)** AB **(c)** C
(d) A **(e)** $\overline{A}C + \overline{B}C$

21. **(a)** $BD + BE + \overline{D}F$ **(b)** $\overline{A}\overline{B}C + \overline{A}\overline{B}D$
(c) B **(d)** $AB + CD$
(e) ABC

23. **(a)** $A\overline{B} + AC + BC$ **(b)** $AC + \overline{B}C$
(c) $AB + AC$

25. **(a)** Domain: A, B, C
Standard SOP: $A\overline{B}C + A\overline{B}\overline{C} + ABC + \overline{A}BC$
(b) Domain: A, B, C
Standard SOP: $ABC + A\overline{B}C + \overline{A}\overline{B}C$
(c) Domain: A, B, C
Standard SOP: $ABC + AB\overline{C} + A\overline{B}C$

27. **(a)** 101 + 100 + 111 + 011
(b) 111 + 101 + 001
(c) 111 + 110 + 101

29. **(a)** $(A + B + C)(A + B + \overline{C})(\overline{A} + \overline{B} + C)(\overline{A} + \overline{B} + C)$
(b) $(A + B + C)(A + \overline{B} + C)(A + \overline{B} + \overline{C})$
$(\overline{A} + B + C) + (\overline{A} + \overline{B} + C)$
(c) $(A + B + C)(A + B + \overline{C})(A + \overline{B} + C)$
$(A + \overline{B} + \overline{C})(\overline{A} + B + C)$

31. **(a)** See Table P–3.

TABLE P–3

A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

31. **(b)** See Table P–4.

TABLE P–4

X	Y	Z	Q
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

33. **(a)** See Table P–5.

TABLE P–5

A	B	C	X
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

33. **(b)** See Table P–6.

TABLE P–6

W	X	Y	Z	Q
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

35. **(a)** See Table P–7.

TABLE P–7

A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

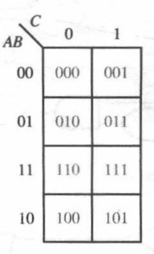
35. **(b)** See Table P–8.

▼TABLE P–8

A	B	C	D	X
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

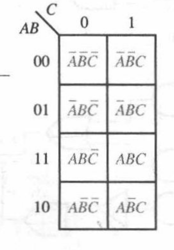
37. See Figure P–15.

▶FIGURE P–15



39. See Figure P–16.

▶FIGURE P–16



41. **(a)** No simplification **(b)** AC
(c) $\overline{D}\overline{F} + E\overline{F}$

43. **(a)** $AB + AC$
(b) $A + BC$
(c) $B\overline{C}D + A\overline{C}D + BC\overline{D} + AC\overline{D}$
(d) $A\overline{B} + CD$

45. $\overline{B} + C$

47. $\overline{A}\overline{B}\overline{C}D + C\overline{D} + BC + A\overline{D}$

49. **(a)** No reduction
(b) $(W + X)(W + \overline{Z})(X + \overline{Y})(\overline{W} + \overline{X} + \overline{Y} + \overline{Z})$

51. $(\overline{A} + B + \overline{D})(A + \overline{C} + D)(A + \overline{B} + C)$
$(B + \overline{C} + \overline{D})(A + \overline{B} + \overline{C} + D)$

53. Minterms: 1, 3, 5, 6, 7

55. See Table P–9.

▼TABLE P–9

Number of 1s	Minterm	ABCD
0	m_0	0000
1	m_1	0001
2	m_5	0101
	m_6	0110
	m_9	1001
	m_{12}	1100

57. See Table P–10.

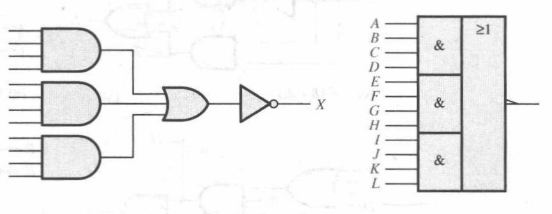
▼TABLE P–10

First Level	Number of 1s in First Level	Second Level
(m_0, m_1) 000x	0	(m_0m_1) 000x
(m_1, m_5) 0x01 (m_1, m_9) x001	1	(m_1, m_5, m_9) xx01

59. $X = \overline{C}D + \overline{A}\,\overline{B}\,\overline{C} + \overline{A}BC\overline{D} + ABC\overline{D}$

Chapter 5

1. See Figure P–17.



▲FIGURE P–17

3. **(a)** $X = ABB$ **(b)** $X = AB + B$
(c) $X = \overline{A} + B$ **(d)** $X = (A + B) + AB$
(e) $X = \overline{ABC}$ **(f)** $X = (A + B)(\overline{B} + C)$

5. **(a)**

A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

5. **(b)**

A	B	X
0	0	0
0	1	1
1	0	0
1	1	1

5. (c)

A	*B*	*X*
0	0	1
0	1	1
1	0	0
1	1	1

5. (d)

A	*B*	*X*
0	0	0
0	1	1
1	0	1
1	1	1

5. (e)

A	*B*	*C*	*X*
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

5. (f)

A	*B*	*C*	*X*
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

7. $X = \overline{A\overline{B} + \overline{A}B} = (\overline{A} + B)(A + \overline{B})$

9. $\overline{ABCD + EFGH}$

11. See Figure P–18.

13. See Figure P–19.

15. $X = AB$

17. **(a)** No simplification

(b) No simplification

(c) $X = A$

(d) $X = \overline{A} + \overline{B} + \overline{C} + EF + \overline{G}$

(e) $X = ABC$

(f) $X = BC\overline{D}E + \overline{A}B\overline{E}FG + BC\overline{E}FG$

19. **(a)** $X = AC + AD + BC + BD$

(b) $X = \overline{A}CD + \overline{B}CD$

(c) $X = ABD + CD + E$

(d) $X = \overline{A} + B + D$

(e) $X = ABD + \overline{C}D + \overline{E}$

(f) $X = \overline{A}\,\overline{C} + \overline{A}\,\overline{D} + \overline{B}\,\overline{C} + \overline{B}\,\overline{D} + \overline{E}\,\overline{G} + \overline{E}\,\overline{H} + \overline{F}\,\overline{G} + \overline{F}\,\overline{H}$

21. See Figure P–20.

23. See Figure P–21.

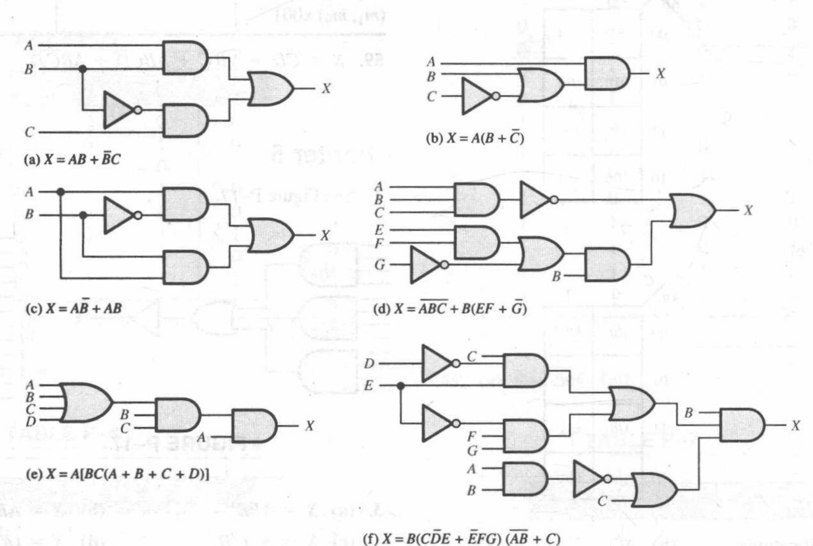
25. See Figure P–22.

27. See Figure P–23.

29. $X = A + \overline{B}$; see Figure P–24.

31. $X = AB\overline{C}$ see Figure P–25.

33. The output pulse width is greater than the specified minimum.



(a) $X = AB + \overline{B}C$

(b) $X = A(B + \overline{C})$

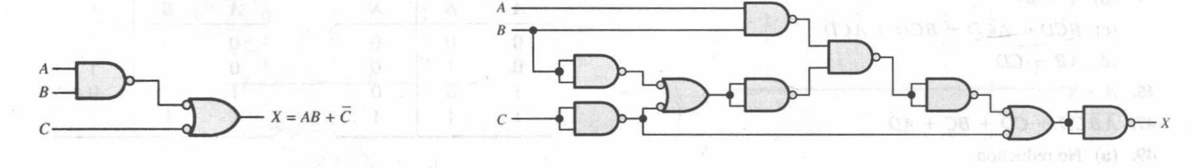
(c) $X = A\overline{B} + AB$

(d) $X = \overline{ABC} + B(EF + \overline{G})$

(e) $X = A[BC(A + B + C + D)]$

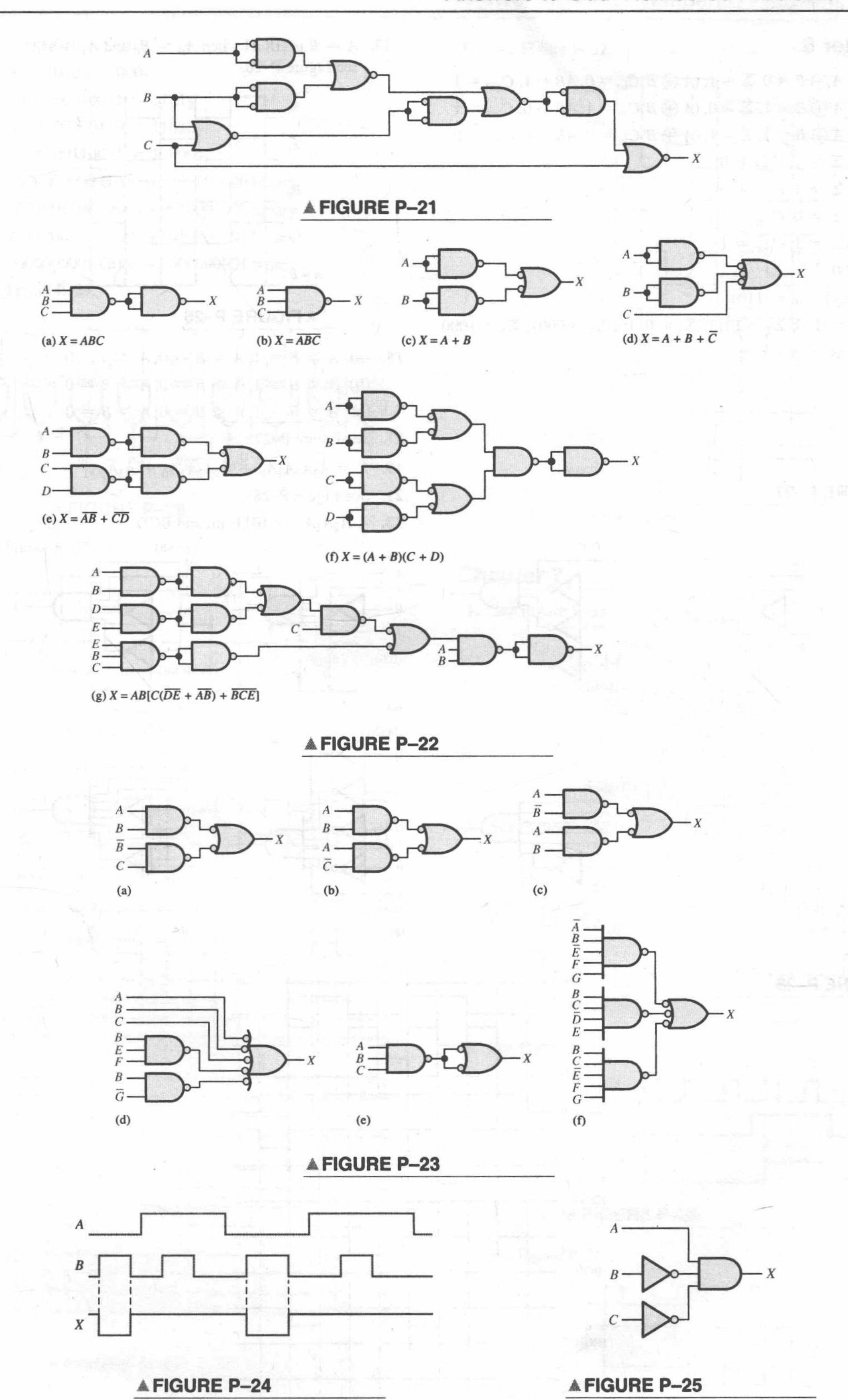
(f) $X = B(C\overline{D}E + \overline{E}FG)\,(\overline{AB} + C)$

▲**FIGURE P–18**



▲**FIGURE P–19**

▲**FIGURE P–20**



▲FIGURE P-21

▲FIGURE P-22

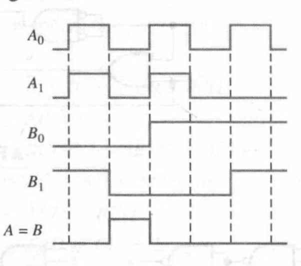
▲FIGURE P-23

▲FIGURE P-24

▲FIGURE P-25

Chapter 6

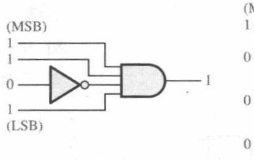
1. **(a)** $A \oplus B = 0, \Sigma = 1, (A \oplus B)C_{\text{in}} = 0, AB = 1, C_{\text{out}} = 1$
   (b) $A \oplus B = 1, \Sigma = 0, (A \oplus B)C_{\text{in}} = 1, AB = 0, C_{\text{out}} = 1$
   (c) $A \oplus B = 1, \Sigma = 1, (A \oplus B)C_{\text{in}} = 0, AB = 0, C_{\text{out}} = 0$
3. **(a)** $\Sigma = 1, C_{\text{out}} = 0$;
   (b) $\Sigma = 1, C_{\text{out}} = 0$;
   (c) $\Sigma = 0, C_{\text{out}} = 1$;
   (d) $\Sigma = 1, C_{\text{out}} = 1$
5. 11100
7. $\Sigma_3\Sigma_2\Sigma_1\Sigma_0 = 1101$
9. $\Sigma_1 = 0110; \Sigma_2 = 1011; \Sigma_3 = 0110; \Sigma_4 = 0001; \Sigma_5 = 1000$
11. 225 ns
13. $A = B$ is HIGH when $A_0 = B_0$ and $A_1 = B_1$; see Figure P–26.

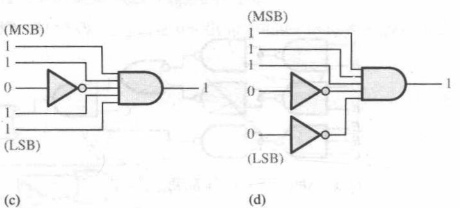


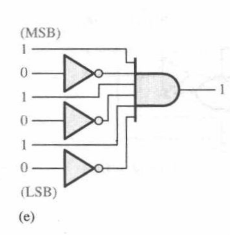
▲ **FIGURE P–26**

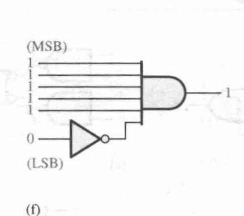
15. **(a)** $A > B = 1; A = B = 0; A < B = 0$
    (b) $A < B = 1; A = B = 0; A > B = 0$
    (c) $A = B = 1; A < B = 0; A > B = 0$
17. See Figure P–27.
19. $X = A_3A_2\bar{A}_1\bar{A}_0 + \bar{A}_3\bar{A}_2\bar{A}_1A_0 + A_3\bar{A}_2A_1$
21. See Figure P–28.
23. $A_3A_2A_1A_0 = 1011$, invalid BCD

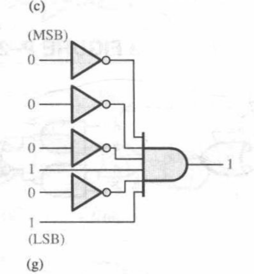
▼ **FIGURE P–27**

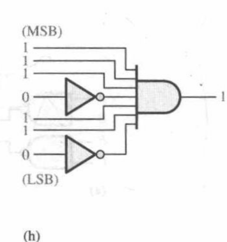




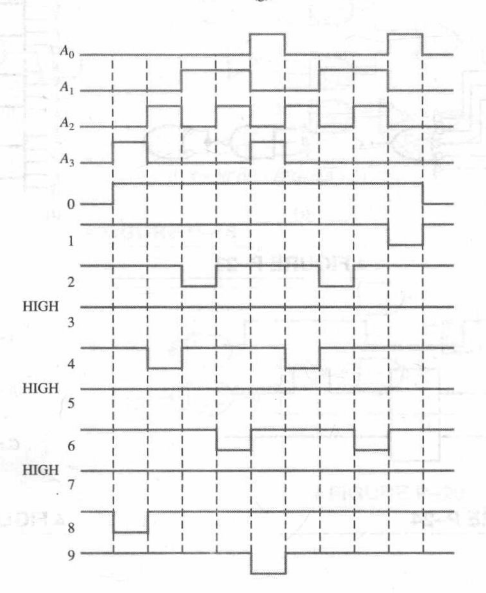








► **FIGURE P–28**



25. (a) $2 = 0010 = 0010_2$

(b) $8 = 1000 = 1000_2$

(c) $13 = 00010011 = 1101_2$

(d) $26 = 00100110 = 11010_2$

(e) $33 = 00110011 = 100001_2$

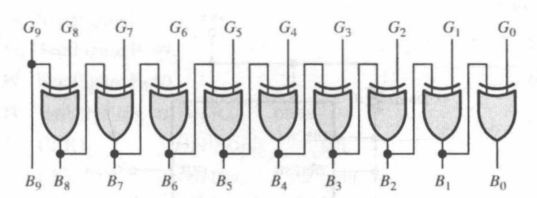
27. (a) 1010000000 Gray → 1100000000 binary

(b) 0011001100 Gray → 0010001000 binary

(c) 1111000111 Gray → 1010000101 binary

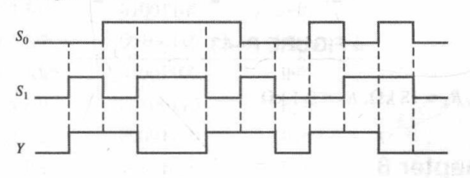
(d) 0000000001 Gray → 0000000001 binary

See Figure P–29.



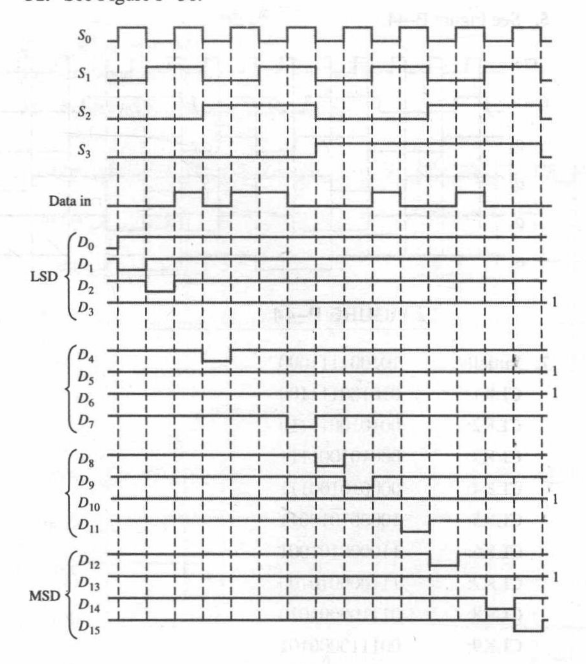
▲ **FIGURE P–29**

29. See Figure P–30.



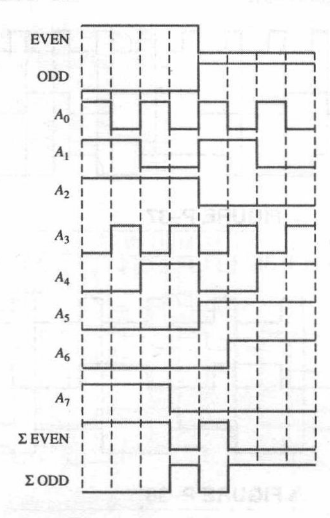
▲ **FIGURE P–30**

31. See Figure P–31.



▲ **FIGURE P–31**

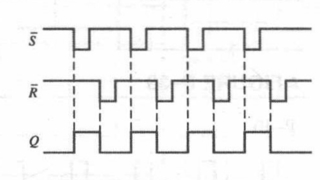
33. See Figure P–32.



▲ **FIGURE P–32**

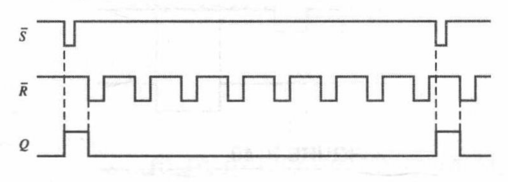
Chapter 7

1. See Figure P–33.



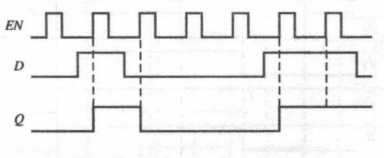
▲ **FIGURE P–33**

3. See Figure P–34.



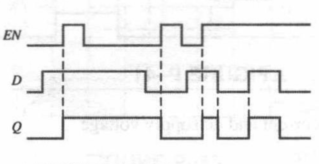
▲ **FIGURE P–34**

5. See Figure P–35.



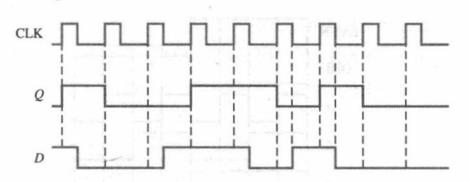
▲ **FIGURE P–35**

7. See Figure P–36.



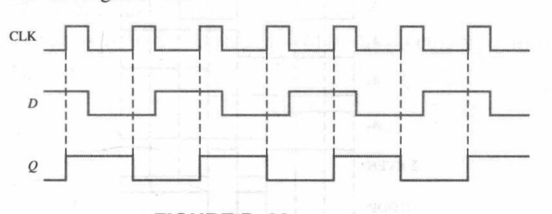
▲ **FIGURE P–36**

9. See Figure P–37.

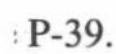


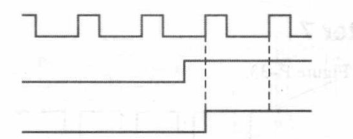
▲ FIGURE P–37

11. See Figure P–38.



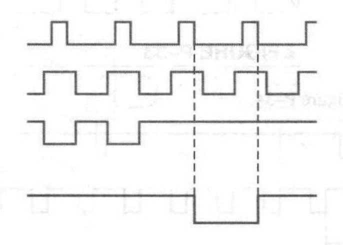
▲ FIGURE P–38


: P-39.



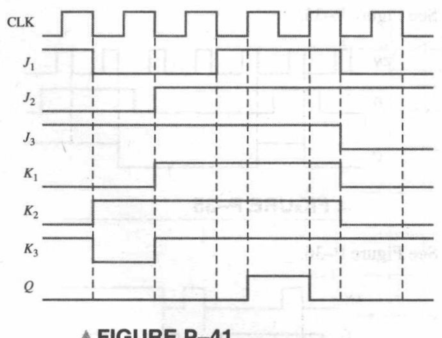
▲ FIGURE P–39

: P–40.



▲ FIGURE P–40

17. See Figure P–41.



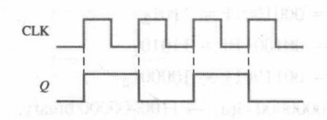
▲ FIGURE P–41

19. Direct current and dc supply voltage

21. 14.9 MHz

23. 150 mA, 750 mW

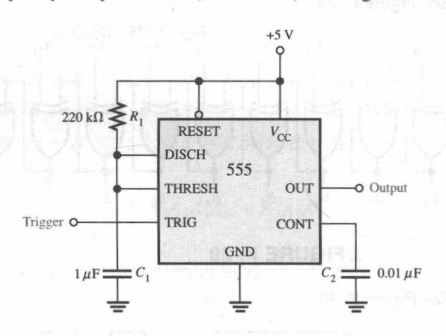
25. divide-by-2; see Figure P–42.



▲ FIGURE P–42

27. 4.62 μs

29. $C_1 = 1\ \mu$F, $R_1 = 227$ kΩ (use 220 kΩ). See Figure P–43.



▲ FIGURE P–43

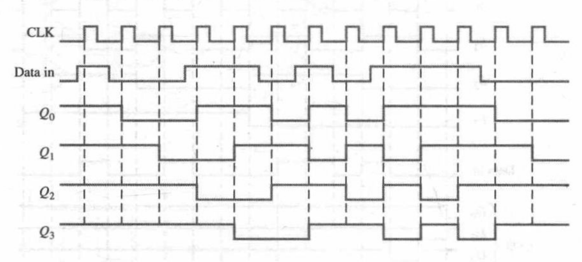
31. $R_1 = 18$ kΩ, $R_2 = 9.1$ kΩ.

Chapter 8

1. Shift registers store binary data.

3. Shift data and store data.

5. See Figure P–44.

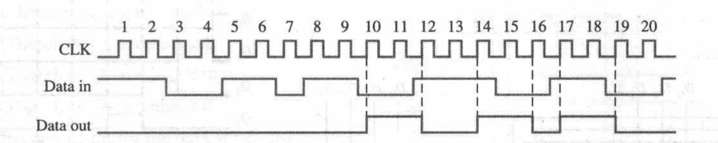


▲ FIGURE P–44

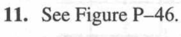
7.

Initially:	101001111000
CLK1:	010100111100
CLK2:	001010011110
CLK3:	000101001111
CLK4:	000010100111
CLK5:	100001010011
CLK6:	110000101001
CLK7:	111000010100
CLK8:	011100001010
CLK9:	001110000101
CLK10:	000111000010
CLK11:	100011100001
CLK12:	110001110000

9. See Figure P–45.



▲ **FIGURE P–45**



11. See Figure P–46.

13. See Figure P–47.

15. See Figure P–48.

17. See Figure P–49.

19. See Figure P–50.

21. Initially (76): 01001100

CLK1:	10011000	left
CLK2:	01001100	right
CLK3:	00100110	right
CLK4:	00010011	right
CLK5:	00100110	left
CLK6:	01001100	left
CLK7:	00100110	right
CLK8:	01001100	left
CLK9:	00100110	right
CLK10:	01001100	left
CLK11:	10011000	left

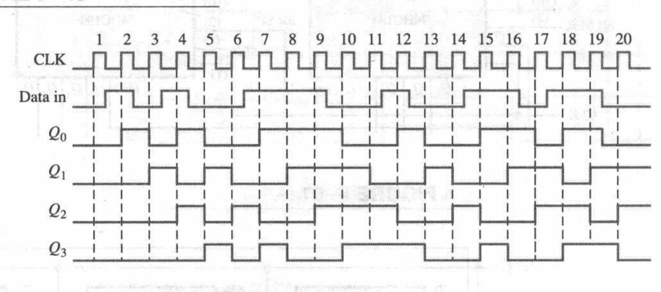
23. See Figure P–51.

25. **(a)** 3 **(b)** 5
(c) 7 **(d)** 8

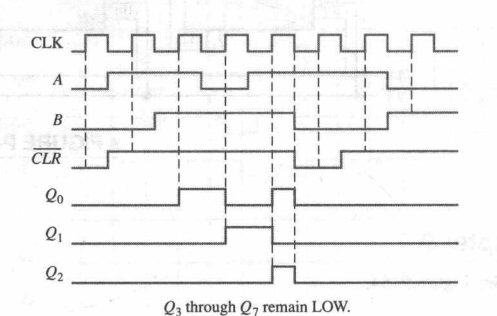
27. See Figure P–52.

29. See Figure P–53.

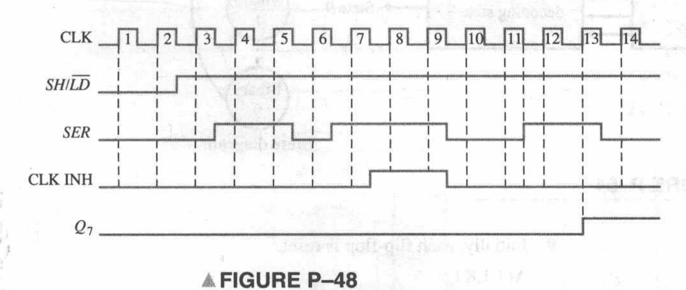
31. An incorrect code may be produced.



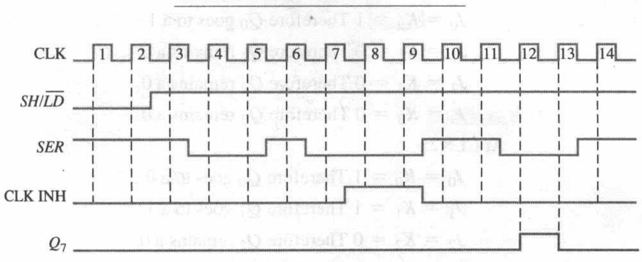
▲ **FIGURE P–46**



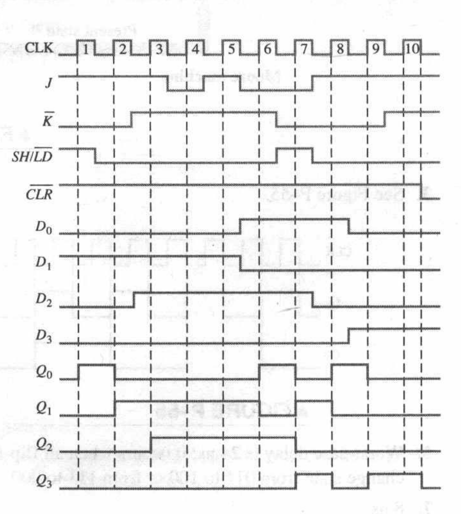
▲ **FIGURE P–47**



▲ **FIGURE P–48**



▲ **FIGURE P–49**



▲ **FIGURE P–50**

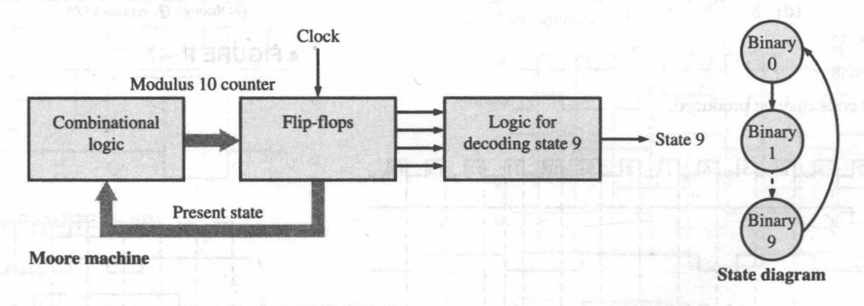
▲ FIGURE P–51

▲ FIGURE P–52

▲ FIGURE P–53

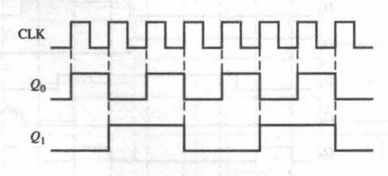
Chapter 9

1. See Figure P–54.



▲ FIGURE P–54

3. See Figure P–55.



▲ FIGURE P–55

5. Worst-case delay is 24 ns; it occurs when all flip-flops change state from 011 to 100 or from 111 to 000.

7. 8 ns

9. Initially, each flip-flop is reset.

   At CLK1:

   $J_0 = K_0 = 1$ Therefore Q_0 goes to a 1.

   $J_1 = K_1 = 0$ Therefore Q_1 remains a 0.

   $J_2 = K_2 = 0$ Therefore Q_2 remains a 0.

   $J_3 = K_3 = 0$ Therefore Q_3 remains a 0.

   At CLK2:

   $J_0 = K_0 = 1$ Therefore Q_0 goes to a 0.

   $J_1 = K_1 = 1$ Therefore Q_1 goes to a 1.

   $J_2 = K_2 = 0$ Therefore Q_2 remains a 0.

   $J_3 = K_3 = 0$ Therefore Q_3 remains a 0.

At CLK3:

$J_0 = K_0 = 1$ Therefore Q_0 goes to a 1.

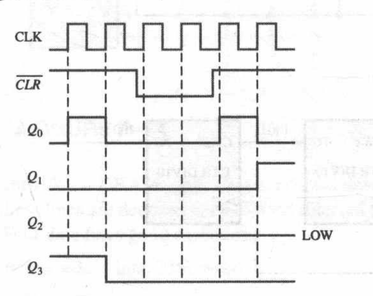
$J_1 = K_1 = 0$ Therefore Q_1 remains a 1.

$J_2 = K_2 = 0$ Therefore Q_2 remains a 0.

$J_3 = K_3 = 0$ Therefore Q_3 remains a 0.

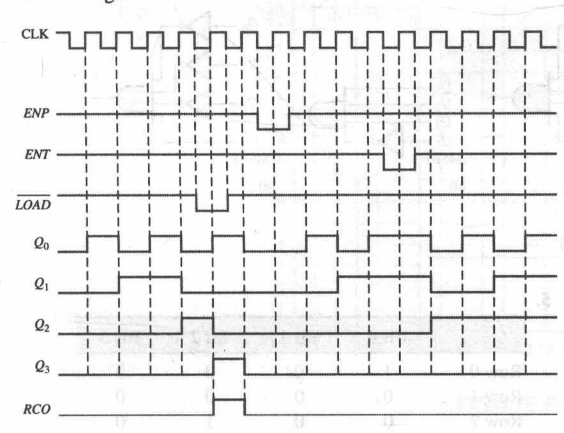
A continuation of this procedure for the next seven clock pulses will show that the counter progresses through the BCD sequence.

11. See Figure P–56.



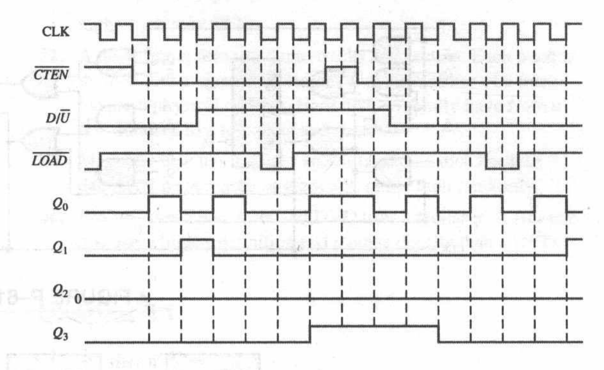
▲FIGURE P–56

13. See Figure P–57.



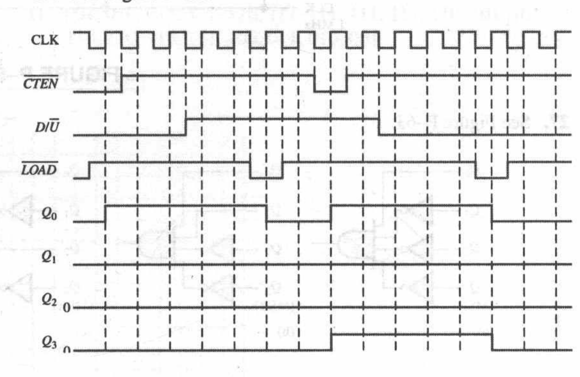
▲FIGURE P–57

15. See Figure P–58.



▲FIGURE P–58

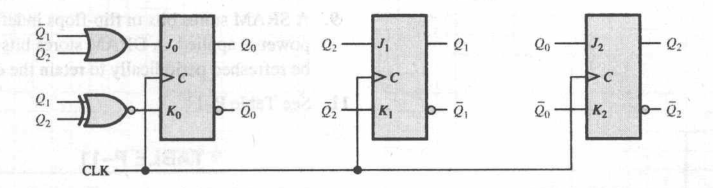
17. See Figure P–59.



▲FIGURE P–59

19. The sequence is 0000, 1111, 1110, 1101, 1010, 0101. The counter "locks up" in the 1010 and 0101 states and alternates between them.

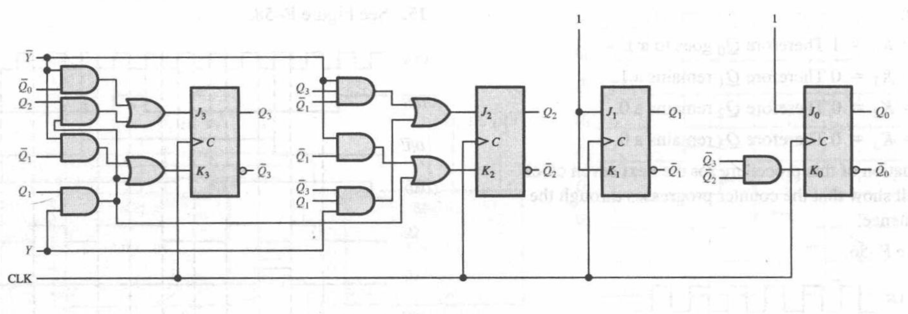
21. See Figure P–60.



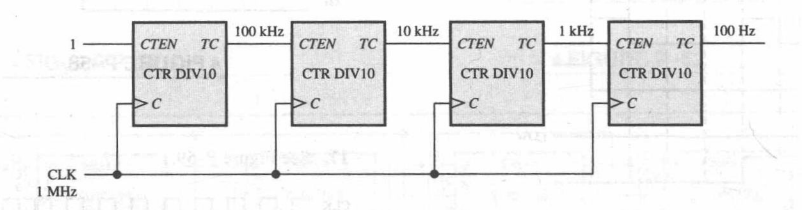
▲FIGURE P–60

23. See Figure P–61.

25. See Figure P–62 for divide-by-10,000. Add one more DIV 10 counter to create a divide-by-100,000.

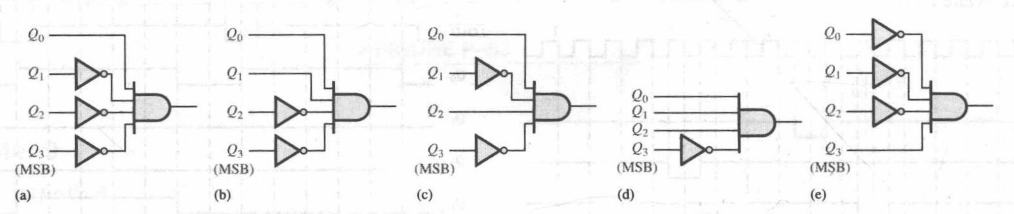


▲FIGURE P–61



▲FIGURE P–62

27. See Figure P–63.



▲FIGURE P–63

29. CLK2, output 0; CLK4, outputs 2, 0; CLK6, output 4; CLK8, outputs 6, 4, 0; CLK10, output 8; CLK12, outputs 10, 8; CLK14, output 12; CLK16, outputs 14, 12, 8

31. A glitch of the AND gate output occurs on the 111 to 000 transition. Eliminate by ANDing $\overline{CLK}$ with counter outputs (strobe) or use Gray code.

33. Hours tens: 0001

Hours units: 0010

Minutes tens: 0000

Minutes units: 0001

Seconds tens: 0000

Seconds units: 0010

35. 68

Chapter 10

1. (a) ROM (b) RAM

3. *Address bus* provides for transfer of address code to memory for accessing any memory location in any order for a read or write operation. *Data bus* provides for transfer of data between the microprocessor and the memory or I/O.

5.

	Bit 0	Bit 1	Bit 2	Bit 3
Row 0	1	0	0	0
Row 1	0	0	0	0
Row 2	0	0	1	0
Row 3	0	0	0	0

7. 512 row × 128 8-bit columns

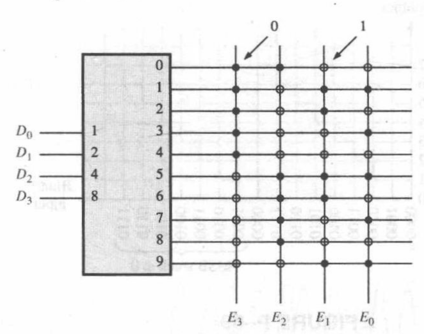
9. A SRAM stores bits in flip-flops indefinitely as long as power is applied. A DRAM stores bits in capacitors that must be refreshed periodically to retain the data.

11. See Table P–11.

▼TABLE P–11

Inputs		Outputs			
A_1	A_0	O_3	O_2	O_1	O_0
0	0	0	1	0	1
0	1	1	0	0	1
1	0	1	1	1	0
1	1	0	0	1	0

13. See Figure P–64.



▲ **FIGURE P–64**

16. Use eight 16k × 4 DRAMs with sixteen address lines. Two of the address lines are decoded to enable the selected memory chips. Four data lines go to each chip.

18. 8 bits, 64k words; 4 bits, 256k words

20. lowest address: $FC0_{16}$

highest address: FFF_{16}

22. A hard disk is formatted into tracks and sectors. Each track is divided into a number of sectors with each sector of a track having a physical address. Hard disks typically have from a few hundred to a few thousand tracks.

24. Magnetic tape has a longer access time than disk because data must be accessed sequentially rather than randomly.

26. Blu-ray uses a blue laser and DVD uses a red laser. A Blu-ray disc has a higher definition and storage density than a DVD.

Chapter 11

1. See Figure P–65.

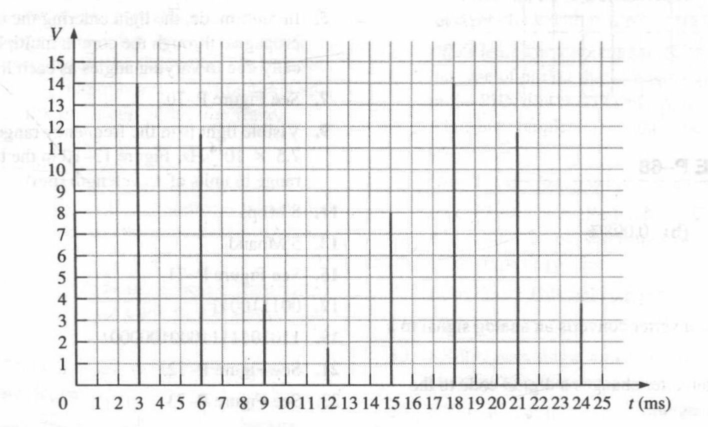
3. 11, 11, 11, 11, 01, 11, 11, 11, 11

5. See Figure P–66.

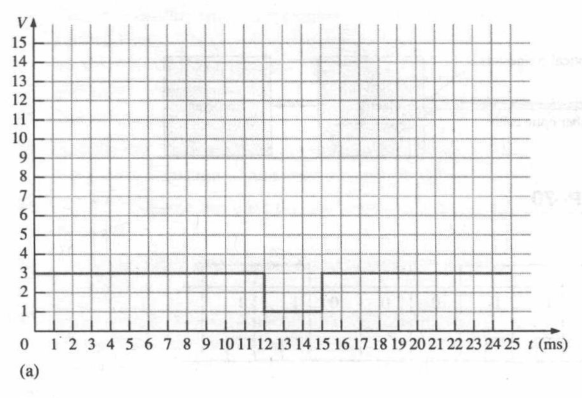
7. 200

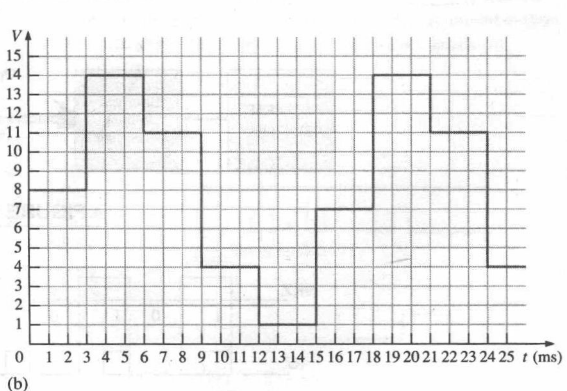
9. −21.4

11. 001, 010, 011, 101, 110, 111, 111, 111, 111, 110, 101, 101, 110, 110, 110, 101, 100, 011, 010, 001



▲ **FIGURE P–65**

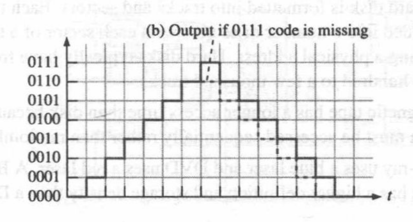




▲ **FIGURE P–66**

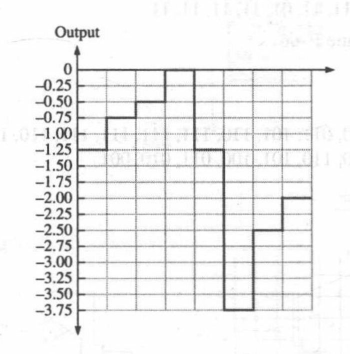
13. 11, 11, 11

15. See Figure P–67.



▲ **FIGURE P–67**

17. See Figure P–68.



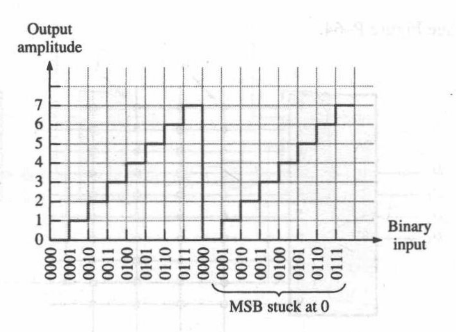
▲ **FIGURE P–68**

19. **(a)** 14.3% **(b)** 0.098%
(c) 0.00038%

21. See Figure P–69.

23. An analog-to-digital converter converts an analog signal to a digital code.

25. A digital-to-analog converter changes a digital code to the corresponding analog signal.



▲ **FIGURE P–69**

Chapter 12

1. A coaxial cable consists of an outer jacket, metallic shield, dielectric, and center conductor.

3. Advantages of fiber optics over electrical transmission media are faster data rates, higher signal capacity (more signals at a time), transmission over longer distances, and not susceptible to EMI.

5. In multimode, the light entering the optical fiber will tend to propagate through the core in multiple rays (modes), basically due to varying angles as each light ray moves along.

7. See Figure P–70.

9. Visible light is in the frequency range 4×10^{14} Hz to 7.5×10^{14} Hz. Figure 13–10 in the textbook shows the range in units of wavelength (nm).

11. 8 Mbps

13. 5 Mbaud

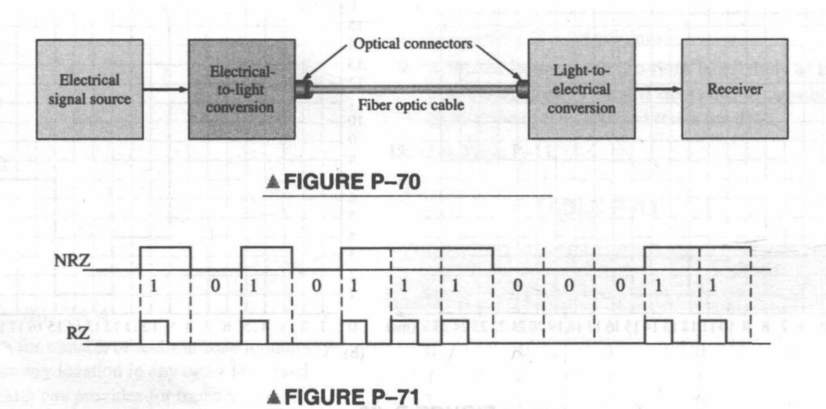
15. See Figure P–71.

17. 001110011

19. 11010111110001000001

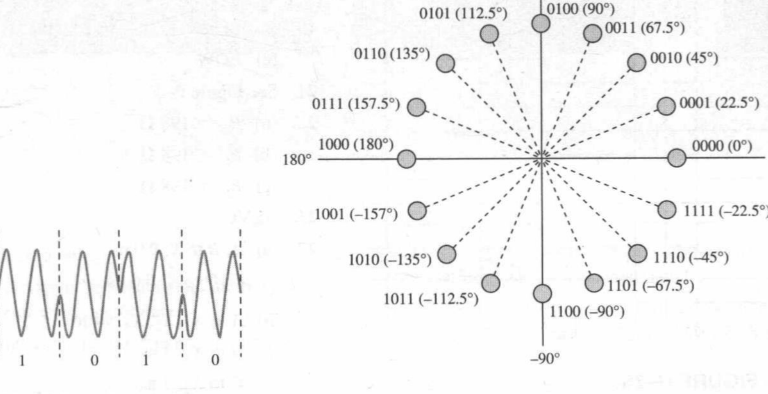
21. See Figure P–72.

23. See Figure P–73.



▲ **FIGURE P–70**

▲ **FIGURE P–71**



▲FIGURE P–72

▲FIGURE P–73

25. In the PWM intersective method, the sawtooth intersects the sinusoidal modulating signal twice during each cycle. The sawtooth is either increasing above the sine wave or decreasing below the sine wave. When the sawtooth is increasing above the sine wave, a low level is generated, and when it is decreasing below the sine wave, a high level is generated. The resulting output is a series of pulses with widths proportional to the amplitude of the sine wave.

27. Data rate = 300 bps

29. Four bits

31. Bit interleaved: A single data bit from a source is transmitted on the channel, followed by a data bit from another source, and so on.

    Byte interleaved: A byte of data from a source is transmitted on the channel, followed by a byte from another source, and so on.

33. In FDM, band-pass filters are used on the receiving end to separate the transmitted signals.

35. *Physical characteristics of a bus:* Number of conductors, length, configuration (serial or parallel), type of connector, number of connector pins, pin configuration.

    Electrical/performance characteristics of a bus: Signal format, signal voltage, clock frequency, transfer speed, bandwidth, data frame format, handshaking protocol, error detection, impedances.

37. 200 MBps and 191 MBps

39. A differential bus provides much higher data rates and longer transmission distances than does a single-ended bus.

41. The PCI-Express bus does not use a shared bus as PCI and PCI-X do.

43. The terms *talker* and *listener* are associated with the IEE-488 bus (GPIB).

45. Three data bytes are transferred because the NDAC line goes HIGH three times, each time indicating that a data byte is accepted.

47. **(a)** SCSI

    (b) USB

    (c) Super speed USB (V 3.0)

49. *Sync Field:* All packets start with a sync (synchronization) field. The sync field consists of 8 bits for low and full speed or 32 bits for high speed and is used to synchronize the receiver clock with that of the transmitter.

    PID Field: The packet identification field is used to identify the type of packet that is being transmitted. There are 4 bits in the PID; however, to ensure it is received correctly, the 4 bits are complemented and repeated, making an 8-bit PID code.

    Data Field: Contains up to 1024 bytes of data.

    CRC Field: Cyclic Redundancy Checks are performed on the data within the packet using from 5 bits to 16 bits, depending on the type of packet.

    EOP Field: This field signals the end of a packet.

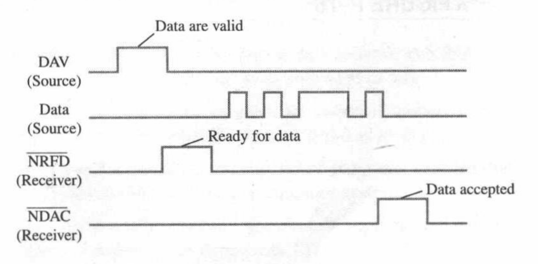
51. 1024 bytes

53. RS-232 uses single-ended transmission. RS-422 uses differential transmission.

55. I^2C is an internal serial bus primarily for connecting ICs on a PC board.

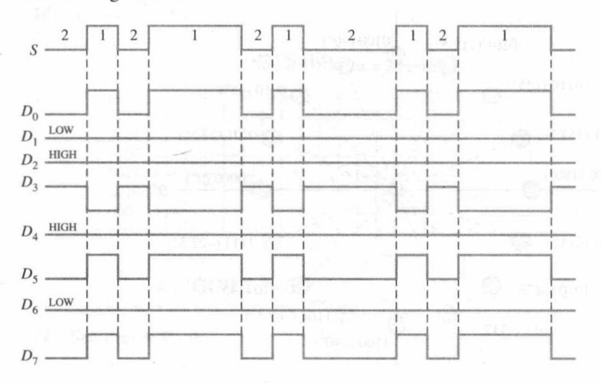
57. Other possible units on an automotive CAN system include wiper control unit, parking control unit, entertainment system unit, tire pressure monitor, seat position unit, heads-up display unit.

59. See Figure P–74.



▲FIGURE P–74

61. See Figure P–75.



▲FIGURE P–75

Chapter 13①

(Chapter 13 is on the website.)

1. No; $V_{OH(min)} < V_{IH(min)}$

3. 0.15 V in HIGH state; 0.25 V in LOW state.

5. Gate *C*

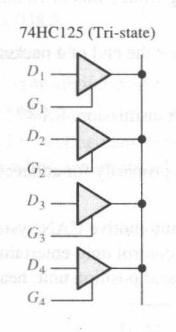
7. 12 ns

9. Gate *C*

11. Yes, G_2

13. **(a)** on **(b)** off
(c) off **(d)** on

15. See Figure P–76 for one possible circuit.



▲FIGURE P–76

17. **(a)** HIGH **(b)** Floating
(c) HIGH **(d)** High-Z

19. **(a)** LOW **(b)** LOW
(c) LOW

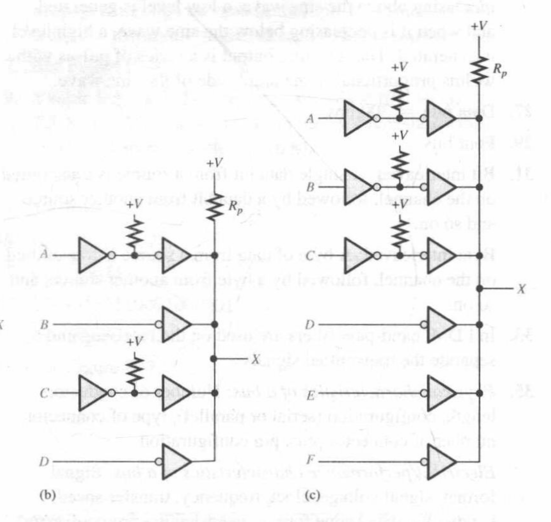
21. See Figure P–77.

23. **(a)** $R_p = 198\ \Omega$
(b) $R_p = 198\ \Omega$
(c) $R_p = 198\ \Omega$

25. ALVC

27. **(a)** *A*, *B* to *X*: 9.9 ns
C, *D* to *X*: 6.6 ns
(b) *A* to X_1, X_2, X_3: 14 ns
B to X_1: 7 ns
C to X_2: 7 ns
D to X_3: 7 ns
(c) *A* to *X*: 11.1 ns
B to *X*: 11.1 ns
C to *X*: 7.4 ns
D to *X*: 7.4 ns

29. ECL operates with nonsaturated BJTs.



▲FIGURE P–77

① Chatper 13指英文原版的在线章节“Integrated Circuit Techonologies”。

Glossary

acceptor A receiving device on a bus.

access time The time from the application of a valid memory address to the appearance of valid output data.

addend In addition, the number that is added to another number called the augend.

adder A logic circuit used to add two binary numbers.

address The location of a given storage cell or group of cells in a memory; a unique memory location containing one byte.

address bus A one-way group of conductors from the microprocessor to a memory, or other external device, on which the address code is sent.

adjacency Characteristic of cells in a Karnaugh map in which there is a single-variable change from one cell to another cell next to it on any of its four sides.

aliasing The effect created when a signal is sampled at less than twice the signal frequency. Aliasing creates unwanted frequencies that interfere with the signal frequency.

alphanumeric Consisting of numerals, letters, and other characters.

ALU Arithmetic logic unit; the key processing element of a microprocessor that performs arithmetic and logic operations.

amplitude In a pulse waveform, the height or maximum value of the pulse as measured from its low level.

analog Being continuous or having continuous values, as opposed to having a set of discrete values.

analog-to-digital (A/D) conversion The process of converting an analog signal to digital form.

analog-to-digital converter (ADC) A device used to convert an analog signal to a sequence of digital codes.

AND A basic logic operation in which a true (HIGH) output occurs only when all the input conditions are true (HIGH).

AND array An array of AND gates consisting of a matrix of programmable interconnections.

AND gate A logic gate that produces a HIGH output only when all of the inputs are HIGH.

ANSI American National Standards Institute.

antifuse A type of PLD nonvolatile programmable link that can be left open or can be shorted once as directed by the program.

architecture The VHDL unit that describes the internal operation of a logic function; the internal functional arrangement of the elements that give a device its particular operating characteristics.

array In a PLD, a matrix formed by rows of product-term lines and columns of input lines with a programmable cell at each junction. In VHDL, an array is an ordered set of individual items called elements with a single identifier name.

ASCII American Standard Code for Information Interchange; the most widely used alphanumeric code.

ASK Amplitude shift keying; a form of modulation in which a digital signal modulates the amplitude of a higher frequency sine wave.

assembler A program that converts English-like mnemonics into machine code.

assembly language A programming language that uses English-like words and has a one-to-one correspondence to machine language.

associative law For addition (ORing) and multiplication (ANDing) of three or more variables, the order in which the variables are grouped makes no difference.

astable Having no stable state. An astable multivibrator oscillates between two quasi-stable states.

asynchronous Having no fixed time relationship; not occurring at the same time.

asynchronous counter A type of counter in which each stage is clocked from the output of the preceding stage.

augend In addition, the number to which the addend is added.

bandwidth The frequency at which a sinusoidal input signal is attenuated to 70.7 percent of its original amplitude.

bank A section of memory within a single memory array (chip).

base One of the three regions in a bipolar junction transistor.

base address The beginning address of a segment of memory.

baud The number of symbols per second in a data transmission.

BCD Binary coded decimal; a digital code in which each of the decimal digits, 0 through 9, is represented by a group of four bits.

BEDO DRAM Burst extended data output dynamic random-access memory.

BiCMOS A family of logic circuits that combines CMOS and bipolar logic.

bidirectional Having two directions. In a bidirectional shift register, the stored data can be shifted right or left.

binary Having two values or states; describes a number system that has a base of two and utilizes 1 and 0 as its digits.

BIOS Basic input/output system; a set of programs in ROM that interfaces the I/O devices in a computer system.

bipolar A class of integrated logic circuits implemented with bipolar transistors; also known as TTL.

bistable Having two stable states. Flip-flops and latches are bistable multivibrators.

bit A binary digit, which can be either a 1 or 0.

bit rate The number of bits per second in a data transmission.

bitstream A series of bits describing a final design that is sent to the target device during programming.

bit time The interval of time occupied by a single bit in a sequence of bits; the period of the clock.

BJT Bipolar junction transistor; a semiconductor device used for switching or amplification. A BJT has two junctions, the base-emitter junction and the base-collector junction.

Blue-ray A disc storage technology that uses a blue laser to achieve more density and definition than a DVD.

Boolean addition In Boolean algebra, the OR operation.

Boolean algebra The mathematics of logic circuits.

Boolean expression A formulation of variables and operators used to express the operation of a logic circuit.

Boolean multiplication In Boolean algebra, the AND operation.

boundary scan A method for internally testing a PLD based on the JTAG standard (IEEE Std. 1149.1).

break point A flag placed within a program source code to stop a program for investigation.

buffer A circuit that prevents loading of an input or output.

bus A set of connections and specifications for the transfer of data among two or more devices.

bus arbitration The process that prevents two sources from using a bus at the same time.

bus contention An adverse condition that could occur if two or more devices try to communicate at the same time on a bus.

bus master Any device that can control and manage the system buses in a computer system.

bus protocol A set of rules that allow two or more devices to communicate through a bus.

byte A group of eight bits.

cache memory A relatively small, high-speed memory that stores the most recently used instructions or data from the larger but slower main memory.

caching The process of copying frequently accessed program instructions from main memory into faster memory to increase processing speed.

capacity The total number of data units (bits, nibbles, bytes, words) that a memory can store.

carry The digit generated when the sum of two binary digits exceeds 1.

carry generation The process of producing an output carry in a full-adder when both input bits are 1s.

carry propagation The process of rippling an input carry to become the output carry in a full-adder when either or both of the input bits are 1s and the input carry is a 1.

cascade To connect "end-to-end" as when several counters are connected from the terminal count output of one counter to the enable input of the next counter.

cascading Connecting two or more similar devices in a manner that expands the capability of one device.

CCD Charge-coupled device; a type of semiconductor memory that stores data in the form of charge packets and is serially accessed.

CD-R CD-Recordable; an optical disk storage device on which data can be stored once.

CD-ROM An optical disk storage device on which data are prestored and can only be read.

CD-RW CD-Rewritable; an optical disk storage on which data can be written and overwritten many times.

cell An area on a Karnaugh map that represents a unique combination of variables in product form; a single storage element in a memory; a fused cross point of a row and column in a PLD.

character A symbol, letter, or numeral.

circuit An arrangement of electrical and/or electronic components interconnected in such a way as to perform a specified function.

CLB Configurable logic block; a unit of logic in an FPGA that is made up of multiple smaller logic modules and a local programmable interconntect that is used to connect logic modules within the CLB.

clear An asynchronous input used to reset a flip-flop (make the Q output 0); to place a register or counter in the state in which it contains all 0s.

clock The basic timing signal in a digital system; a periodic waveform used to synchronize operation.

cloud storage A remote network of servers that is connected to a user device through the Internet.

CMOS Complementary metal oxide semiconductor; a class of integrated logic circuits that is implemented with a type of field-effect transistor.

coaxial cable A type of data transmission media in which a shielded conductor is used to minimize EMI.

code A set of bits arranged in a unique pattern and used to represent such information as numbers, letters, and other symbols; in VHDL, program statements.

codec A combined coder and decoder.

collector One of the three regions in a bipolar transistor.

combinational logic A combination of logic gates interconnected to produce a specified Boolean function with no storage or memory capability; sometimes called *combinatorial logic*.

commutative law In addition (ORing) and multiplication (ANDing) of two variables, the order in which the variables are ORed or ANDed makes no difference.

comparator A digital circuit that compares the magnitudes of two quantities and produces an output indicating the relationship of the quantities.

compiler An application program in development software packages that controls the design flow process and translates source code into object code in a format that can be logically tested or downloaded to a target device.

complement The inverse or opposite of a number; in Boolean algebra, the inverse function, expressed with a bar over the variable. The complement of a 1 is a 0, and vice versa.

component A VHDL feature that can be used to predefine the logic function for multiple use throughout a program or programs.

contiguous Joined together.

control bus A set of conductive paths that connects the CPU to other parts of the computer to coordinate its operations and to communicate with external devices.

controller An instrument that can specify each of the other instruments on the bus as either a talker or a listener for the purpose of data transfer.

control unit The portion within the microprocessor that provides the timing and control signals for getting data into and out of the microprocessor and for synchronizing the execution of instructions.

counter A digital circuit capable of counting electronic events, such as pulses, by progressing through a sequence of binary states.

CPLD A complex programmable logic device that consists basically of multiple SPLD arrays with programmable interconnections.

CPU Central processing unit; the main part of a computer responsible for control and processing of data; the core of a DSP that processes the program instructions.

cross-assembler A program that translates an assembly language program for one type of microprocessor to an assembly language for another type of microprocessor.

crosstalk The presence of an unwanted signal via an accidental coupling.

current sinking The action of a circuit in which it accepts current into its output from a load.

current sourcing The action of a circuit in which it sends current out of its output and into a load.

cyclic redundancy check (CRC) A type of error detection code.

data Information in numeric, alphabetic, or other form.

data bus A bidirectional set of conductive paths on which data or instruction codes are transferred into a microprocessor or on which the result of an operation is sent out from the microprocessor.

data center A facility that houses a cloud storage system.

data selector A circuit that selects data from several inputs one at a time in a sequence and places them on the output; also called a multiplexer.

data sheet A document that specifies parameter values and operating conditions for an integrated circuit or other device.

DCE Data communications equipment.

DDR Double data rate.

DDR SDRAM Double data rate, synchronous dynamic random-access memory.

decade Characterized by ten states or values.

decade counter A digital counter having ten states.

decimal Describes a number system with a base of ten.

decode The process of interpreting coded information; changing data in a coded form into a more common form; a stage of the DSP pipeline operation in which instructions are assigned to functional units and are decoded.

decoder A digital circuit (device) that converts coded information into another (familiar) or noncoded form.

decrement To decrease the binary state of a counter by one.

delta modulation A method of analog-to-digital conversion using a 1-bit quantization process.

design flow The process or sequence of operations carried out to program a target device.

D flip-flop A type of bistable multivibrator in which the output assumes the state of the D input on the triggering edge of a clock pulse.

demultiplexer (demux) A circuit (digital device) that switches digital data from one input line to several output lines in a specified time sequence.

dependency notation A notational system for logic symbols that specifies input and output relationships, thus fully defining a given function; an integral part of ANSI/IEEE Std. 91-1984.

difference The result of a subtraction.

differential operation A bus operation that uses two wires for data (one for data and one for the complement of the data) and one wire for ground.

digit A symbol used to express a quantity.

digital Related to digits or discrete quantities; having a set of discrete values as opposed to continuous values.

digital signal processor (DSP) A special type of microprocessor that processes data in real time.

digital-to-analog (D/A) conversion The process of converting a sequence of digital codes to an analog form.

digital-to-analog converter (DAC) A device in which information in digital form is converted to analog form.

DIMM Dual in-line memory module.

diode A semiconductor device that conducts current in only one direction.

DIP Dual in-line package; a type of IC package whose leads must pass through holes to the other side of a PC board.

distributive law The law that states that ORing several variables and then ANDing the result with a single variable is equivalent to ANDing the single variable with each of the several variables and then ORing the product.

dividend In a division operation, the quantity that is being divided.

divisor In a division operation, the quantity that is divided into the dividend.

DLT Digital linear tape; a type of magnetic tape format.

DMA Direct memory access; a method to directly interface a peripheral device to memory without using the CPU for control.

domain All of the variables in a Boolean expression.

"Don't care" A combination of input literals that cannot occur and can be used as a 1 or a 0 on a Karnaugh map for simplification.

downloading A design flow process in which the logic design is transferred from software to hardware.

drain One of the terminals of a field-effect transistor.

DRAM Dynamic random-access memory; a type of semiconductor memory that uses capacitors as the storage elements and is a volatile, read/write memory.

DSP core The central processing unit of a digital system processor.

DTE Data terminal equipment.

duty cycle The ratio of pulse width to period expressed as a percentage.

DVD-ROM Digital versatile disk-ROM; also known as digital video disk-ROM; a type of optical storage device on which data is prestored with a much higher capacity than a CD-ROM.

dynamic memory A type of semiconductor memory having capacitive storage cells that lose stored data over a period of time and, therefore, must be refreshed.

ECL Emitter-coupled logic; a class of integrated logic circuits that are implemented with nonsaturating bipolar junction transistors.

E^2CMOS Electrically erasable CMOS (EECMOS); the circuit technology used for the reprogrammable cells in a PLD.

edge-triggered flip-flop A type of flip-flop in which the data are entered and appear on the output on the same clock edge.

EDIF Electronic design interchange format; a standard form of netlist.

EDO DRAM Extended data output dynamic random-access memory.

EEPROM Electrically erasable programmable read-only memory; a type of nonvolatile PLD reprogrammable link based on electrically-erasable programmable read-only memory cells and can be turned on or off repeatedly by programming.

8 mm A type of magnetic tape format.

elasticity The ability of a cloud storage system to deal with variations in the amount of data being transferred without service interruptions.

electromagnetic waves Related to the electromagnetic spectrum, which includes radio waves, microwaves, infrared, visible, ultraviolet, X-rays, and gamma rays.

embedded system Generally, a single-purpose system, such as a processor, built into a larger system for the purpose of controlling the system.

EMI Electromagnetic interference.

emitter One of the three regions in a bipolar junction transistor.

enable To activate or put into an operational mode; an input on a logic circuit that permits its operation.

encoder A digital circuit (device) that converts information to a coded form.

entity The VHDL unit that describes the inputs and outputs of a logic function.

EPROM Erasable programmable read-only memory; A type of PLD nonvolatile programmable link based on electrically programmable read-only memory cells and can be turned either on or off once with programming.

error detection The process of detecting bit errors in a digital code.

even parity The condition of having an even number of 1s in every group of bits.

exception Any software event that requires special handling by the processor.

exclusive-NOR (XNOR)gate A logic gate that produces a LOW only when the two inputs are at opposite levels.

exclusive-OR (XOR) A basic logic operation in which a HIGH occurs when the two inputs are at opposite levels.

exclusive-OR (XOR) gate A logic gate that produces a HIGH only when the two inputs are at opposite levels.

execute A CPU process in which an instruction is carried out; a stage of the DSP pipeline operation in which the decoded instructions are carried out.

exponent The part of a floating-point number that represents the number of places that the decimal point (or binary point) is to be moved.

fall time The time interval between the 90% point and the 10% point on the negative-going edge of a pulse.

fan-out The number of equivalent gate inputs of the same family series that a logic gate can drive.

FDM Frequency division multiplexing; a broadband technique in which the total bandwidth available to a system is divided into frequency sub-bands and information is sent in analog form.

feedback The output voltage or a portion of it that is connected back to the input of a circuit.

FET Field-effect transistor.

fetch A CPU process in which an instruction is obtained from the memory; a stage of the DSP pipeline operation in which an instruction is obtained from the program memory.

FIFO First in–first out memory.

Firewire A high-speed external serial bus standard developed by Apple Inc. and used in high-speed communications and real-time data transfer, also known as IEEE-1394.

firmware Small fixed programs and/or data structures that internally control various electronic devices; usually stored in ROM.

fixed-function logic A category of digital integrated circuits having functions that cannot be altered.

flag A bit that indicates the result of an arithmetic or logic operation or is used to alter an operation.

flash A type of PLD nonvolatile reprogrammable link technology based on a single transistor cell.

flash ADC A simultaneous analog-to-digital converter.

flash memory A nonvolatile read/write random-access semiconductor memory in which data is stored as charge on the floating gate of a certain FET.

flip-flop A basic storage circuit that can store only one bit at a time; a synchronous bistable device.

floating-point number A number representation based on scientific notation in which the number consists of an exponent and a mantissa.

forward bias A voltage polarity condition that allows a semiconductor *pn* junction in a transistor or diode to conduct current.

FPGA Field-programmable gate array; a programmable logic device that uses the LUT as the basic logic elements and generally employs either antifuse or SRAM-based process technology.

FPM DRAM Fast page mode dynamic random-access memory.

frequency (f) The number of pulses in one second for a periodic waveform. The unit of frequency is the hertz.

FSK Frequency shift keying; a form of modulation in which a digital signal modulates the frequency of a higher frequency sine wave.

full-adder A digital circuit that adds two bits and an input carry to produce a sum and an output carry.

full-duplex A connection in which the data flow both ways simultaneously in the same channel.

functional simulation A software process that tests the logical or functional operation of a design.

fuse A type of PLD nonvolatile programmable link that can be left shorted or can be opened once as directed by the program; also called a fusible link.

GAL Generic array logic; a reprogrammable type of SPLD that is similar to a PAL except that it uses a reprogrammable process technology, such as EEPROM (E^2 CMOS), instead of fuses.

gate A logic circuit that performs a basic logic operation, such as AND or OR; one of the three terminals of a field-effect transistor.

glitch A voltage or current spike of short duration, usually unintentionally produced and unwanted.

graphic (schematic) entry A method of entering a logic design into software by graphically creating a logic diagram (schematic) on a design screen.

GPIB General-purpose interface bus based on the IEEE 488 standard.

Gray code An unweighted digital code characterized by a single bit change between adjacent code numbers in a sequence.

half-adder A digital circuit that adds two bits and produces a sum and an output carry. It cannot handle input carries.

half-duplex A connection in which the data flow both ways but not at the same time in the same channel.

Hamming code An error detection and correction code used in data transmission.

handshaking The process of signal interchange by which two digital devices or systems jointly establish communication.

hard core A fixed portion of logic in an FPGA that is put in by the manufacturer to provide a specific function.

hard disk A magnetic disk storage device; typically, a stack of two or more rigid disks enclosed in a sealed housing.

hardware The circuitry and physical components of a computer system (as opposed to the instructions called software).

HDL Hardware description language; a language used for describing a logic design using software.

hexadecimal Describes a number system with a base of 16.

high-level language A type of computer language closest to human language that is a level above assembly language.

high-Z The high-impedance state of a tri-state circuit in which the output is effectively disconnected from the rest of the circuit.

hit rate The percentage of memory accesses that find the requested data in the given level of memory.

hold time The time interval required for the control levels to remain on the inputs to a flip-flop after the triggering edge of the clock in order to reliably activate the device.

HPIB Hewlett-Packard interface bus; same as GPIB (general-purpose interface bus).

hysteresis A characteristic of a threshold-triggered circuit, such as the Schmitt trigger, where the device turns on and off at different input levels.

IEEE Institute of Electrical and Electronics Engineers.

IEEE 488 bus Same as GPIB (general-purpose interface bus); a standard parallel bus used widely for test and measurement interfacing.

IEEE 1394 A serial bus for high-speed data transfer; also known as FireWire.

I^2L Integrated injection logic; an IC technology.

implementation The software process where the logic structures described by the netlist are mapped into the structure of the target device; the physical realization of a conceptual design.

increment To increase the binary state of a counter by one.

inhibit To prevent the passage of a signal from one point to another.

input The signal or line going into a circuit; a signal that controls the operation of a circuit.

input/output (I/O) A terminal of a device that can be used as either an input or as an output.

instruction One step in a computer program; a unit of information that tells the CPU what to do.

in-system programming (ISP) A method for programming SPLDs after they are installed on a printed circuit board and operating in a system.

integer A whole number.

integrated circuit (IC) A type of circuit in which all of the components are integrated on a single chip of semiconductive material of very small size.

intellectual property (IP) Designs owned by the manufacturer of programmable logic devices or other products.

interfacing The process of making two or more electronic devices or systems operationally compatible with each other so that they function properly together.

interrupt Any hardware event that requires special handling by the processor, an event that causes the current process to be temporarily stopped while a service routine is run.

inversion The conversion of a HIGH level to a LOW level or vice versa; also called complementation.

inverter A NOT circuit; a circuit that changes a HIGH to a LOW or vice versa.

I/O port Input/output port; the interface between an internal bus and a peripheral.

IP Instruction pointer; a special register within the CPU that holds the offset address of the next instruction to be executed.

ISA bus Industry standard architecture bus; an internal parallel bus standard.

J-K flip-flop A type of flip-flop that can operate in the SET, RESET, no-change, and toggle modes.

Johnson counter A type of register in which a specific prestored pattern of 1s and 0s is shifted through the stages, creating a unique sequence of bit patterns.

JTAG Joint test action group; the IEEE Std. 1149.1 standard interface for in-system programming.

junction The boundary between an *n* region and a *p* region in a BJT.

Karnaugh map An arrangement of cells representing the combinations of literals in a Boolean expression and used for a systematic simplification of the expression.

LAB Logic array block; an SPLD array in a CPLD.

latch A bistable digital circuit used for storing a bit.

latency The time between the request for data and the delivery of the data to the user.

latency period The time it takes for the desired sector to spin under the head once the head is positioned over the desired track of a magnetic hard disk.

LCC Leadless ceramic chip; an SMT package that has metallic contacts molded into its body.

LCD Liquid crystal display.

leading edge The first transition of a pulse.

least significant bit (LSB) Generally, the right-most bit in a binary whole number or code.

LED Light-emitting diode.

LIFO Last in–first out memory, memory stack.

listener An instrument capable of receiving data on a GPIB (general-purpose interface bus) when it is addressed by the computer.

literal A variable or the complement of a variable.

load To enter data into a shift register.

loading The effect of the multiple inputs degrading the voltage or timing specifications of an output.

local bus An internal bus that connects the microprocessor to the cache memory, the main memory, the coprocessor, and the PCI bus controller.

local interconnect A set of lines that allows interconnections among the eight logic elements in a logic array block without using the row and column interconnects.

logic In digital electronics, the decision-making capability of gate circuits, in which a HIGH represents a true statement and a LOW represents a false one.

logic array block (LAB) A group of macrocells that can be interconnected with other LABs or to other I/Os using a programmable interconnect array; also called a function block.

logic element The smallest section of logic in an FPGA that typically contains an LUT, associated logic, and a flip-flop.

look-ahead carry A method of binary addition whereby carries from preceding adder stages are anticipated, thus eliminating carry propagation delays.

LSI Large-scale integration; a level of fixed-function IC complexity in which there are from more than 100 to 10,000 equivalent gates per chip.

LUT Look-up table; a type of memory that can be programmed to produce SOP functions.

machine code The basic binary instructions understood by the processor.

machine language Computer instructions written in binary code that are understood by a computer; the lowest level of programming language.

macrocell An SOP logic array with combinational and registered outputs; part of a PAL or GAL that generally consists of one OR gate and some associated output logic. Multiple interconnected macrocells form a CPLD.

magneto-optical disk A storage device that uses electro-magnetism and a laser beam to read and write data.

magnitude The size or value of a quantity.

main memory Memory used by computer systems to store the bulk of programs and associated data.

Manchester encoding A method of encoding called biphase in which a 1 is represented by a positive-going transition and a 0 is represented by a negative-going transition.

mantissa The magnitude of a floating-point number.

Mealy state machine A state machine in which the outputs depend on both the internal present state and on the inputs.

mechatronics Interdisciplinary field that comprises both mechanical and electronic components.

memory The portion of a computer or other system that stores binary data.

memory array An array of memory cells arranged in rows and columns.

memory hierarchy The arrangement of various memory elements within a computer architecture to achieve maximum performance.

memory latency The time required to access a memory.

MFLOPS Million floating-point operations per second.

microcontroller A semiconductor device that combines a microprocessor, memory, and various hardware peripherals on a single IC.

microprocessor A large-scale digital integrated circuit device that can be programmed with a series of instructions to perform specified functions on data.

minimization The process that results in an SOP or POS Boolean expression that contains the fewest possible terms with the fewest possible literals per term.

minterm A product of literals in which each input variable appears exactly once.

minuend The number from which another number is subtracted.

MIPS Million instructions per second.

miss A failed attempt by the processor to read or write a block of data in a given level of memory.

MMACS Million multiply/accumulates per second.

MMU Memory management unit; a device responsible for handling accesses to memory requested by the CPU.

mnemonic An English-like instruction that is converted by an assembler into a machine code for use by a processor.

modem A modulator/demodulator for interfacing digital devices to analog transmission systems such as telephone lines.

modulation The process of altering a parameter of a higher frequency signal proportional to the amplitude of a lower frequency information-carrying signal.

modulus The number of unique states through which a counter will sequence.

monostable Having only one stable state. A monostable multivibrator, commonly called a one-shot, produces a single pulse in response to a triggering input.

monotonic The characteristic of a DAC defined by the absence of any incorrect step reversals; one type of digital-to-analog linearity.

Moore state machine A state machine in which the outputs depend only on the internal present state.

MOS Metal-oxide semiconductor; a type of transistor technology.

MOSFET Metal-oxide semiconductor field-effect transistor.

most significant bit (MSB) The left-most bit in a binary whole number or code.

MSI Medium-scale integration; a level of fixed-function IC complexity in which there are from 10 to 100 equivalent gates per chip.

multicore processor A microprocessor chip with more than one processor.

multimode The characteristic of an optical fiber in which the light is propagated in multiple rays.

multiplexer (mux) A circuit (digital device) that switches digital data from several input lines onto a single output line in a specified time sequence.

multiplicand The number that is being multiplied by another number.

multiplier The number that multiplies the multiplicand.

multiprocessing A data-processing technique that uses multiple processors to multitask or run multiple programs.

multitasking A technique by which a processor runs multiple programs concurrently.

multitenancy The property of a cloud storage system that allows multiple users to share the same software applications, hardware, and data storage mechanism without seeing each other's data.

multithreading The process of executing different parts of a program, called threads, simultaneously.

multivibrator A class of digital circuits in which the output is connected back to the input (an arrangement called feedback) to produce either two stable states, one stable state, or no stable states, depending on the configuration.

NAND gate A logic circuit in which a LOW output occurs only if all the inputs are HIGH.

negative-AND An equivalent NOR gate operation in which the HIGH is the active input when all inputs are LOW.

negative-OR An equivalent NAND gate operation in which the HIGH is the active input when one or more of the inputs are LOW.

netlist A detailed listing of information necessary to describe a circuit, such as types of elements, inputs, and outputs, and all interconnections.

nibble A group of four bits.

NMOS An *n*-channel metal-oxide semiconductor.

node A common connection point in a circuit in which a gate output is connected to one or more gate inputs.

noise immunity The ability of a circuit to reject unwanted signals.

noise margin The difference between the maximum LOW output of a gate and the maximum acceptable LOW input of an equivalent gate; the difference between the minimum HIGH output of a gate and the minimum HIGH input of an equivalent gate; the amount by which the actual signal level exceeds the minimum acceptable level for an error-free transmission.

nonvolatile A term that describes a memory that can retain stored data when the power is removed.

NOR gate A logic gate in which the output is LOW when any or all of the inputs are HIGH.

NOT A basic logic operation that performs inversions.

NRZ Nonreturn to zero; a type of data format in which the signal level remains at one (1) for successive 1s.

numeric Related to numbers.

Nyquist frequency The highest signal frequency that can be sampled at a specified sampling frequency; a frequency equal to or less than half the sampling frequency.

object program A machine language translation of a high-level source program.

octal Describes a number system with a base of eight.

odd parity The condition of having an odd number of 1s in every group of bits.

offset address The distance in number of bytes of a physical address from the base address.

OLMC Output logic macrocell; the part of a GAL that can be programmed for either combinational or registered outputs; a block of logic in a GAL that contains a fixed OR gate and other logic for handling inputs and/or outputs.

one-shot A monostable multivibrator.

op-code Operation code; the code representing a particular microprocessor instruction; a mnemonic.

open-collector A type of output in a logic circuit in which the collector of the output transistor is left disconnected from any internal circuitry and is available for external connection; normally used for driving higher-current or higher-voltage loads.

operand The object to be manipulated by the instruction.

operating system The software that controls the computer system and oversees the execution of application software.

operational amplifier (op-amp) A device with two differential inputs that has very high gain, very high input impedance, and very low output impedance.

optical fiber A type of data transmission media used for transmitting light signals.

optical jukebox A type of auxiliary storage for very large amounts of data.

OR A basic logic operation in which a true (HIGH) output occurs when one or more of the input conditions are true (HIGH).

OR gate A logic gate that produces a HIGH output when one or more inputs are HIGH.

oscillator An electronic circuit that is based on the principle of regenerative feedback and produces a repetitive output waveform; a signal source.

OTP One-time programmable.

output The signal or line coming out of a circuit.

overflow The condition that occurs when the number of bits in a sum exceeds the number of bits in each of the numbers added.

packet A formatted block of digital data.

PAL Programmable array logic; a type of one-programmable SPLD that consists of a programmable array of AND gates that connects to a fixed array of OR gates.

PAM Pulse amplitude modulation; a method of modulation in which the height or amplitude of the pulses are varied according to the modulating analog signal, and each pulse represents a value of amplitude of the analog signal.

parallel In digital systems, data occurring simultaneously on several lines; the transfer or processing of several bits simultaneously.

parallel bus A bus that consists of multiple conductors and carries several data bits simultaneously, one on each conductor.

parity In relation to binary codes, the condition of evenness or oddness of the number of 1s in a code group.

parity bit A bit attached to each group of information bits to make the total number of 1s odd or even for every group of bits.

PCI bus An internal synchronous bus for interconnecting chips, expansion boards, and processor/memory subsystems.

PCI-Express Also designated as PCIe or PCI-E. This bus differs from the PCI and PCI-X buses in that it does not use a shared bus.

PCI-X A high-performance enhancement of the PCI bus that is backward compatible with PCI.

PCM Pulse code modulation; A method of modulation that involves sampling of an analog signal amplitude at regular intervals and converting the sampled values to a digital code.

period (*T*) The time required for a periodic waveform to repeat itself.

periodic Describes a waveform that repeats itself at a fixed interval.

peripheral A device or instrument that provides communication with a computer or provides auxiliary services or functions for the computer.

physical address The actual location of a data unit in memory.

PIC Programable interrupt controller; handles the interrupts on a priority basis.

pipeline As applied to memories, an implementation that allows a read or write operation to be initiated before the previous operation is completed; part of the DSP architecture that allows multiple instructions to be processed simultaneously.

pipelining A technique where the processor begins executing the next instruction before the previous instruction has been completed.

PLA Programmable logic array; an SPLD with programmable AND and OR arrays.

platform FPGA An FPGA that contains either or both hard core and soft core embedded processors and other functions.

PLCC Plastic leaded chip carrier; an SMT package whose leads are turned up under its body in a J-type shape.

PLD Programmable logic device; an integrated circuit that can be programmed with any specified logic function.

PMOS A *p*-channel metal-oxide semiconductor.

pointer The contents of a register (or registers) that contain an address.

polling The process of checking a series of peripheral devices to determine if any require service from the CPU.

port A physical interface on a computer through which data are passed to or from peripherals.

positive logic The system of representing a binary 1 with a HIGH and a binary 0 with a LOW.

power dissipation The product of the dc supply voltage and the dc supply current in an electronic circuit; the amount of power required by a circuit.

PPM Pulse position modulation; a method of modulation in which the position of each pulse relative to a reference or timing signal is varied proportional to the amplitude of the modulating signal waveform.

prefetching The process of executing instructions at the same time as other instructions are "fetched," eliminating idle time; also called pipelining.

preset An asynchronous input used to set a flip-flop (make the *Q* output 1).

priority encoder An encoder in which only the highest value input digit is encoded and any other active input is ignored.

probe An accessory used to connect a voltage to the input of an oscilloscope or other instrument.

processes Instances of a computer program that are being executed.

product The result of a multiplication.

product-of-sums (POS) A form of Boolean expression that is basically the ANDing of ORed terms.

product term The Boolean product of two or more literals equivalent to an AND operation.

program A list of computer instructions arranged to achieve a specific result; software.

programmable interconnect array (PIA) An array consisting of conductors that run throughout the CPLD chip and to which connections from the macrocells in each LAB can be made.

programmable logic A category of digital integrated circuits capable of being programmed to perform specified functions.

PROM Programmable read-only semiconductor memory; an SPLD with a fixed AND array and programmable OR array; used as a memory device and normally not as a logic circuit device.

propagation delay time The time interval between the occurrence of an input transition and the occurrence of the corresponding output transition in a logic circuit.

protocol A standardized set of software regulations, requirements, and procedures that control and regulate the transmission, processing, and exchange of data among devices.

pseudo-operation An instruction to the assembler (as opposed to a processor).

PSK Phase shift keying; a form of modulation in which a digital signal modulates the phase of a higher frequency sine wave.

pull-up resistor A resistor with one end connected to the dc supply voltage used to keep a given point in a circuit HIGH when in the inactive state.

pulse A sudden change from one level to another, followed after a time, called the pulse width, by a sudden change back to the original level.

pulse width (t_w) The time interval between the 50% points of the leading and trailing edges of the pulse; the duration of the pulse.

PWM Pulse width modulation; a method of modulation in which the width or duration of the pulses and duty cycle are varied according to the modulating analog signal, and each pulse width represents an amplitude value of the analog signal.

QAM Quadrature amplitude modulation; a form of modulation that uses a combination of PSK and amplitude modulation to send information.

QIC Quarter-inch cassette; a type of magnetic tape.

quantization The process whereby a binary code is assigned to each sampled value during analog-to-digital conversion.

queue A high-speed memory that stores instructions or data.

quotient The result of a division.

race A condition in a logic network in which the difference in propagation times through two or more signal paths in the network can produce an erroneous output.

RAM Random-access memory; a volatile read/write semiconductor memory.

rank A group of chips that make up a memory module that stores data in units such as words or bytes.

read The process of retrieving data from a memory.

real mode Operation of an Intel processor in a manner to emulate the 8086's 1 MB of memory.

record length The number of samples (data points) that an oscilloscope can capture and store.

recycle To undergo transition (as in a counter) from the final or terminal state back to the initial state.

refresh To renew the contents of a dynamic memory by recharging the capacitor storage cells.

register A digital circuit capable of storing and shifting binary information; typically used as a temporary storage device.

register array A set of temporary storage locations within the microprocessor for keeping data and addresses that need to be accessed quickly by the program.

registered A CPLD macrocell output configuration where the output comes from a flip-flop.

relocatable code A program that can be moved anywhere within the memory space without changing the basic code.

remainder The amount left over after a division.

RESET The state of a flip-flop or latch when the output is 0; the action of producing a RESET state.

resolution The number of bits used to digitally represent a sampled value.

reverse bias A voltage polarity condition that prevents a *pn* junction of a transistor or diode from conducting current.

ring counter A register in which a certain pattern of 1s and 0s is continuously recirculated.

ripple carry A method of binary addition in which the output carry from each adder becomes the input carry of the next higher-order adder.

ripple counter An asynchronous counter.

rise time The time required for the positive-going edge of a pulse to go from 10% of its full value to 90% of its full value.

ROM Read-only semiconductor memory, accessed randomly; also referred to as mask-ROM.

RS-232 A bus standard, also known as EIA-232, used in industrial and telecommunication applications as well as scientific instrumentation, but largely replaced by USB in computer applications.

RS-422 A bus standard for differential data transmission.

RS-423 A bus standard for single-ended data transmission.

RS-485 A bus standard for differential data transmission.

RZ Return to zero; a type of data format in which the signal level goes to or remains at zero after each data bit.

sampling The process of taking a sufficient number of discrete values at points on a waveform that will define the shape of the waveform.

sampling rate The rate at which the analog-to-digital converter (ADC) in an oscilloscope is clocked to digitize the incoming signal.

SAS Serial attached SCSI.

scalability The ability of a cloud storage system to handle increasing amounts of data in a smooth manner. The ability of a cloud storage system to improve the movement of data when additional resources are added.

schematic (graphic) entry A method of placing a logic design into software using schematic symbols.

Schottky A specific type of transistor-transistor logic circuit technology.

SCSI Small computer system interface.

SDRAM Synchronous dynamic random-access memory.

seek time The time for the read/write head in a hard drive to position itself over the desired track for a read operation.

segment A 64k block of memory.

sequential circuit A digital circuit whose logic states follow a specified time sequence.

serial Having one element following another, as in a serial transfer of bits; occurring, as pulses, in sequence rather than simultaneously.

serial bus A bus that carries data bits sequentially one at a time on a single conductor.

server Any computerized process that shares a resource with one or more clients. A computer and software with a large memory capacity that responds to requests across a network to provide file storage and access as well as services such as file sharing.

SET The state of a flip-flop or latch when the output is 1; the action of producing a SET state.

set-up time The time interval required for the control levels to be on the inputs to a digital circuit, such as a flip-flop, prior to the triggering edge of clock pulse.

shared bus A bus, such as PCI, that is shared by multiple devices.

signal A type of VHDL object that holds data.

signal-to-noise ratio (SNR) A measure of the signal strength relative to background noise, usually expressed in decibels (dB).

signal tracing A troubleshooting technique in which waveforms are observed in a step-by-step manner beginning at the input and working toward the output or vice versa. At each point the observed waveform is compared with the correct signal for that point.

sign bit The left-most bit of a binary number that designates whether the number is positive (0) or negative (1).

SIMM Single-in-line memory module.

simplex A connection in which data flows in only one direction from the sender (transmitter) to the receiver.

single-ended operation A bus operation that uses one wire for data and one wire for ground.

single mode The characteristic of an optical fiber in which the light tends to propagate in a single beam or ray.

SMT Surface-mount technology; an IC package technique in which the packages are smaller than DIPs and are mounted on the printed surface of the PC board.

soft core A portion of logic in an FPGA; similar to hard core except it has some programmable features.

software Computer programs; programs that instruct a computer what to do in order to carry out a given set of tasks.

software interrupt An instruction that invokes an interrupt service routine.

SOIC Small-outline integrated circuit; an SMT package that resembles a small DIP but has its leads bent out in a "gull-wing" shape.

source A sending device of a bus; one of the terminals of a field-effect transistor.

source program A program written in either assembly or high-level language.

speed-power product A performance parameter that is the product of the propagation delay time and the power dissipation in a digital circuit.

SPI Serial-to-peripheral interface bus; a synchronous serial communications bus that uses four wires for communication between a "master" device and a "slave" device.

SPLD Simple programmable logic device; an array of AND gates and OR gates that can be programmed to achieve specified logic functions. Four types are PROM, PLA, PAL, and GAL.

SRAM Static random-access memory; a type of PLD volatile reprogrammable link based on static random-access memory cells and can be turned on or off repeatedly with programming.

SSI Small-scale integration; a level of fixed-function IC complexity in which there are up to 10 equivalent gates per chip.

SSOP Shrink small-outline package.

stage One storage element (flip-flop) in a register.

state diagram A graphic depiction of a sequence of states or values.

state machine A logic system or circuit exhibiting a sequence of states conditioned by internal logic and external inputs; any sequential circuit exhibiting a specified sequence of states. Two types of state machine are Moore and Mealy.

static memory A volatile semiconductor memory that uses flip-flops as the storage cells and is capable of retaining data without refreshing.

storage The capability of a digital device to retain bits; the process of retaining digital data for later use.

STP Shielded twisted pair; a type of data transmission medium.

string A contiguous sequence of bytes or words.

strobing A process of using a pulse to sample the occurrence of an event at a specified time in relation to the event.

subroutine A series of instructions that can be assembled together and used repeatedly by a program but programmed only once.

subtracter A logic circuit used to subtract two binary numbers.

subtrahend The number that is being subtracted from the minuend.

sum The result when two or more numbers are added together.

sum-of-products (SOP) A form of Boolean expression that is basically the ORing of ANDed terms.

sum term The Boolean sum of two or more literals equivalent to an OR operation.

synchronous A condition that describes signals or systems that are aligned or synchronized with each other in terms of timed events, two or more systems that have the same timing signal.

synchronous counter A type of counter in which each stage is clocked by the same pulse.

synthesis The software process where the design is translated into a netlist.

system bus The interconnecting paths in a computer system including the address bus, data bus and control bus.

talker An instrument capable of transmitting data on a GPIB (general-purpose interface bus).

tape library A type of auxiliary storage for very large amounts of data.

target device A PLD mounted on a programming fixture or development board into which a software logic design is to be downloaded; the programmable logic device that is being programmed.

TDM Time division multiplexing; a technique in which data from several sources are interleaved on a time basis and sent on a single communication channel or data link.

terminal count The final state in a counter's sequence.

text entry A method of entering a logic design into software using a hardware description language (HDL).

throughput The average speed with which a program is executed.

timer A circuit that can be used as a one-shot or as an oscillator; a circuit that produces a fixed time interval output.

timing diagram A graph of digital waveforms showing the proper time relationship of two or more waveforms and how each waveform changes in relation to the others.

timing simulation A software process that uses information on propagation delays and netlist data to test both the logical operation and the worst-case timing of a design.

toggle The action of a flip-flop when it changes state on each clock pulse.

totem-pole A type of output in TTL circuits.

trailing edge The second transition of a pulse.

transistor A semiconductor device exhibiting current and/or voltage gain. When used as a switching device, it approximates an open or closed switch.

trigger A pulse used to initiate a change in the state of a logic circuit.

tri-state A type of output in logic circuits that exhibits three states: HIGH, LOW, and high-Z; also known as 3-state.

tri-state buffer A circuit used to interface one device to another to prevent loading.

troubleshooting The technique of systematically identifying, isolating, and correcting a fault in a circuit or system.

truth table A table showing the inputs and corresponding output level of a logic circuit.

TTL Transistor-transistor logic; a class of integrated logic circuit that uses bipolar junction transistors. Also called *bipolar*.

ULSI Ultra large-scale integration; a level of IC complexity in which there are more than 100,000 equivalent gates per chip.

unit load A measure of fan-out. One gate input represents a unit load to the output of a gate within the same IC family.

universal gate Either a NAND gate or a NOR gate. The term *universal* refers to the property of a gate that permits any logic function to be implemented by that gate or by a combination of gates of that kind.

universal shift register A register that has both serial and parallel input and output capability.

up/down counter A counter that can progress in either direction through a certain sequence.

USB Universal serial bus; an external serial bus standard.

UTP Unshielded twisted pair; a type of data transmission medium.

UV EPROM Ultraviolet erasable programmable ROM.

variable symbol used to represent an action, a condition, or data that can have a value of 1 or 0, usually designated by an italic letter or word.

VHDL A standard hardware description language; IEEE Std. 1076-1993.

VLSI Very large-scale integration; a level of IC complexity in which there are from more than 10,000 to 100,000 equivalent gates per chip.

volatile The characteristic of a programmable logic device that loses programmed data when power is turned off.

wait state A system bus delay equal to one processor clock cycle. Wait states are used to ensure that the system bus timing satisfies the address, data, and control timing specifications of a system.

weight The value of a digit in a number based on its position in the number.

word A group of bits or bytes that acts as a single entity that can be stored in one memory location; two bytes.

word capacity The number of words that a memory can store.

word length The number of bits in a word.

WORM Write once-read many; a type of optical storage device.

write The process of storing data in a memory.

zero suppression The process of blanking out leading or trailing zeros in a digital display.